UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Detecção de falhas em sistema para Ambiente de Vida Assistida

Airton Antônio de Jesus Junior

SÃO CRISTÓVÃO/SE
JUNHO/2017

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Airton Antônio de Jesus Junior

Detecção de falhas em sistema para Ambiente de Vida Assistida

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Tarcísio da Rocha
Coorientador: Prof. Dr. Edward David Moreno Ordonez

SÃO CRISTÓVÃO/SE
JUNHO/2017

Airton Antônio de Jesus Junior

Detecção de falhas em sistema para Ambiente de Vida Assistida

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal de Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Prof. Dr. Tarcísio, da Rocha, Presidente
Universidade Federal de Sergipe (UFS)

Profª. Drª. Edilayne Meneses Salgueiro, Examinadora interna
Universidade Federal de Sergipe (UFS)

Prof. Dr. Neumar Malheiros, Examinador externo
Universidade Federal de Lavras (UFLA)

DEDICATÓRIA

Dedico este trabalho a Deus,
às minhas amadas esposa Thaíse e mãe Dejanira,
aos meus familiares, professores e amigos pelo
incentivo e apoio sempre oferecidos.

RESUMO

Ambiente de Vida Assistida (AVA) é, essencialmente, uma proposta científica e tecnológica que visa auxiliar a manutenção da vida e o bem-estar de pessoas idosas ou com necessidades especiais. Para tanto, utiliza-se um sistema de controle que integra e coordena dispositivos inteligentes: braceletes, roupas, camas, quartos etc., com vistas a ampliar a autonomia dos indivíduos na execução das mais simples atividades diárias bem como prover remotamente serviços de saúde e de assistência social. Como então garantir, neste contexto, que o sistema não esteja interpretando uma situação equivocada da realidade do meio ou do próprio paciente? Ou ainda, como prevenir respostas inadequadas ou omissões irremediáveis? Apenas com o cumprimento eficaz de requisitos qualitativos, tais como técnicas de detecção de erros e tolerância a falhas poder-se-á minimizar tais circunstâncias inaceitáveis para preservação da saúde do indivíduo assistido. Neste diapasão, o presente estudo visa levantar o contexto de identificação de falhas em sistemas distribuídos assíncronos de modo a projetar, desenvolver e, em seguida, avaliar a eficácia do módulo de detecção de erros em sistema para controle de AVA por meio do emprego de tecnologias abertas (*open hardware/software*). A metodologia empregada para avaliação deste trabalho será a abordagem Goal Question Metric (GQM). Diante dos resultados encontrados identificou-se a necessidade de modificar um algoritmo de detector de erros originalmente proposto para redes móveis auto organizáveis (MANETs) de modo a torná-lo mais apropriado para o contexto de sistema para Ambiente de Vida Assistida.

Palavras-chave: Ambiente de Vida Assistida, Tolerância a Falhas, Detecção de erros e tecnologias abertas (*open hardware/software*).

SUMÁRIO

1. Introdução....13
1.1. Objetivos....15
1.2. Organização do Trabalho....16
2. Revisão Bibliográfica....17
2.1. Ambiente de Vida Assistida....17
2.2. Sistema para Ambiente de Vida Assistida....18
2.3. Dependabilidade em sistema para Ambiente de Vida Assistida....21
2.3.1. Requisitos de dependabilidade em sistema para Ambiente de Vida Assistida...22
2.3.2. OpenHAB, plataforma para sistema tolerante a falhas em AVA....26
2.3.3. Tolerância a falhas em sistemas para Ambiente de Vida Assistida....32
2.3.3.1. Detecção de erros em sistemas distribuídos....33
2.3.3.2. Detector de erros não-confiável em sistemas assíncronos....35
2.3.3.3. Detector de erros em sistemas para Ambiente de Vida Assistida....38
2.3.3.4. Módulo de tolerância a falhas em sistema para AVA....48
2.4. Trabalhos correlatos....49
3. Metodologia....51
3.1. Materiais e Métodos....51
3.1.1. Definição do objetivo do experimento....51
3.1.2. Planejamento do experimento....52
3.1.2.1. Formulação de hipóteses....52
3.1.2.2. Variáveis independentes....53
3.1.2.3. Variável dependente....57
3.1.2.4. Tratamento....58
3.1.3. Projeto do experimento....58
3.1.3.1. Instrumentação....59
3.1.4. Operação do experimento....59
3.1.4.1. Preparação....59
3.1.4.2. Execução....60
3.1.4.2.1. Coleta de dados....64
3.1.4.2.2. Validação dos dados....64
4. Protótipo de Ambiente de Vida Assistida....65

5. Resultados 68
5.1. Análise e interpretação dos dados 68
5.2. Ameaças à validade 72
6. Conclusão 73

Lista de Figuras

Figura 1 : Representação de um AVA (LLORET, CANOVAS, *et al.*, 2015).....18
Figura 2 : Modelo conceitual de um sistema para AVA (TAZARI, FURFARI, *et al.*, 2012) .19
Figura 3 : Arquitetura de referência em Ambiente de Vida Assistida (NEHMER, BECKER, *et al.*, 2006).....20
Figura 4 : Arquitetura do OpenHAB (OPENHAB, 2017).....28
Figura 5 : Comunicação interna do OpenHAB (OPENHAB, 2017).....29
Figura 6 : Esquema de funcionamento do broker publisher-subscriber Mosquitto (SANTOS, 2017).....30
Figura 7 : Publicação de dados sensoriais via MQTT (SANTOS, 2017).....30
Figura 8 : Subscrição de comandos OpenHAB via MQTT (SANTOS, 2017).....31
Figura 9 : Suposto cenário de falta indevida de consenso do detector de erros da classe $\diamond S^M$. 44
Figura 10 : Cenário de propagação de suspeita defasada (0) em colisão com engano atualizado (1) em nó .10. 47
Figura 11 : Processo de construção de um espaço AVA (FURFARI, TAZARI, & EISEMBERG, 2011). 54
Figura 12 : Raspberry Pi Zero (EVERPI, 2015). 55
Figura 13 : ESP8266 NodeMCU (THOMSEN, 2016). 56
Figura 14 : Protótipo de Ambiente de Vida Assistida.....65
Figura 15 : Detalhe do quarto do protótipo de Ambiente de Vida Assistida.....66
Figura 16 : Detalhe da sala e cozinha do protótipo de Ambiente de Vida Assistida 66
Figura 17 : Detalhe da instalação dos dispositivos inteligentes no protótipo de Ambiente de Vida Assistida 66
Figura 18 : Visão geral dos *clusters* de computadores de placa única para execução do experimento.....67
Figura 19 : Detalhe de um c*luster* de computadores de placa única para execução do experimento.....67

Lista de Tabelas

Tabela 1 : Avaliação de requisitos de tolerância a falhas em sistemas para AVA (ANTONINO, SCHNEIDER, *et al.*, 2011). 25
Tabela 2 : Classes de detector de erro não-confiável (CHANDRA e TOUEG, 1996). 38
Tabela 3 : Comparação dos estudos correlatos frente a proposta do corrente estudo.....50

Lista de Quadros

Quadro 1 : Estatística descritiva da variável falsa suspeita de parada (percMistakes) em implementação original (Algoritmo 1) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e *switch* central – Fase 1 68
Quadro 2 : Estatística descritiva da variável falsa suspeita de parada (percMistakes) em implementação modificada (Algoritmo 2) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e *switch* central – Fase 2.....69
Quadro 3 : Teste de normalidade da Fase 1 (versão original – Algoritmo 1).....70
Quadro 4 : Teste de normalidade da Fase 2 (versão modificada – Algoritmo 2) 70
Quadro 5 : Teste U de Mann-Whitney sobre as amostras independentes coletas na Fase 1 e 2 do experimento.....70
Quadro 6 : Detalhe do Teste U de Mann-Whitney sobre as amostras independentes coletas na Fase 1 e 2 do experimento 71

Lista de Gráficos

Gráfico 1 : Transição demográfica no Brasil (2000-2100) (UNITED NATIONS. DEPARTMENT OF ECONOMIC, 2015).....13
Gráfico 2 : Apuração do percentual de falsa suspeita de parada em implementação original (Algoritmo 1) sob o tratamento inicial do experimento (falha pseudoaleatória em apenas um nó) – Fase 1 61
Gráfico 3 : Apuração de percentual de falsa suspeita de parada em implementação modificada (Algoritmo 2) sob o tratamento inicial do experimento (falha pseudoaleatória em apenas um nó) – Fase 2 62
Gráfico 4 : Apuração do percentual de falsa suspeita de parada sob implementação original (Algoritmo 1) sob novo tratamento (falha pseudoaleatória em nó e switch) – Fase 1.....63
Gráfico 5 : Apuração do percentual de falsa suspeita de parada sob implementação modificada (Algoritmo 2) sob novo tratamento (falha pseudoaleatória em nó e switch) – Fase 2.....63

Lista de Algoritmos

Algoritmo 1 : Algoritmo original para implementação de detector de erros da classe $\diamond S^M$ (GREVE, SENS, *et al.*, 2011). 41
Algoritmo 2 : Alterações no Algoritmo 1 para desenvolvimento de módulo detector de falhas em sistema para AVA 48

1. Introdução

O Brasil insere-se no grupo de países que vivencia uma transição demográfica acelerada, devido ao fenômeno da queda acentuada na taxa de fecundidade desde o início de 1960. Este fato vem provocando alterações demográficas por faixa etária, reduzindo a quantidade de crianças e jovens e aumentando a quantidade de adultos e idosos (Gráfico 1). A partir deste fenômeno espera-se que o país experimente um rápido envelhecimento populacional, com importantes implicações para indivíduos, família e sociedade (ERVATTI, BORGES e JARDIM, 2015).

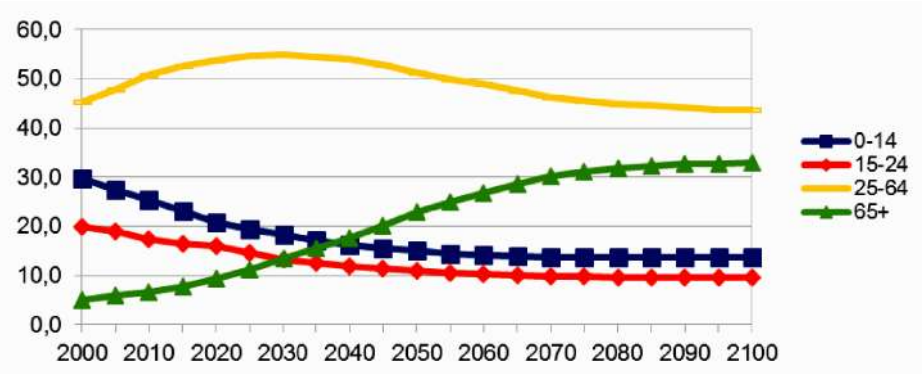


Gráfico 1 : Transição demográfica no Brasil (2000-2100) (UNITED NATIONS. DEPARTMENT OF ECONOMIC, 2015).

A realidade da transição demográfica traz intrinsecamente consigo outro fato que é o da transição epidemiológica. O pesquisador Omram (1996) descreve este processo como sendo a evolução progressiva do perfil de elevada mortalidade originária inicialmente de doenças infecciosas para outra onde predominam os óbitos ocasionados por doenças cardiovasculares, neoplasias, causas externas e outras doenças consideradas crônico-degenerativas. De tais doenças destacam-se a hipertensão arterial e a diabetes tipo 2. Sendo que a primeira acomete no mínimo 25% da população brasileira acima de 34 anos e a segunda 7,6% das pessoas acima de 30 anos (DA SILVA FILHO, 2016).

Embora a transição epidemiológica no Brasil esteja ocorrendo de maneira diferenciada nas diferentes regiões do país, pode-se afirmar que o aumento da mortalidade pelas doenças crônico-degenerativas vem ocorrendo, principalmente, a partir da segunda metade do século XX (FELICIANO e DE MORAES, 1999).

Dada a constatação do progressivo envelhecimento das pessoas no país, deduz-se que o

sistema de saúde deveria se amoldar as tais demandas deste novo contingente populacional (idosos). Todavia, o país ainda vem enfrentando grandes desafios nos quesitos básicos de saúde, tais como controle da mortalidade infantil e de doenças transmissíveis (por exemplo: zika vírus, dengue e febre amarela), quanto mais se encontra preparado para prevenir e tratar as doenças crônico-degenerativas e suas complicações (CHAIMOWICZ, 1997).

Ressalta-se, no entanto, que nem sempre as políticas públicas acompanham as demandas demográficas, vez que outros fatores – políticos, sociais e institucionais – são considerados num contexto de disputas de interesses frente ao Estado e aos recursos orçamentários (ERVATTI, BORGES e JARDIM, 2015).

Dessa forma, diante do cenário político-demográfico nacional que se delineia aliado às questões de enormes desigualdades econômicas e sociais, os idosos passam a não encontrar o amparo adequado no sistema público de saúde e previdência, e, por isso, desenvolvem os efeitos das doenças crônico-degenerativas, adquirem incapacidades, perdem autonomia e, consequentemente, qualidade de vida (CHAIMOWICZ, 1997). O desejável, todavia, para uma sociedade humanamente civilizada seria motivar os idosos a se manterem socialmente ativos e plenamente independentes (NAKAGAWA, ANTONINO, *et al.*, 2013).

Diante de tal contexto, vislumbra-se a necessidade da sociedade, em especial a comunidade técnico-científica, vislumbrar soluções funcionais e acessíveis que possam promover a autonomia e, por conseguinte, a melhora na condição de saúde e no bem-estar de pessoas idosas ou dotadas de cuidados especiais. Neste sentido, uma das propostas estimuladas pela comunidade europeia é o desenvolvimento de processos e tecnologias capazes de implementar o conceito de Ambiente de Vida Assistida, AVA (*Ambient Assisted Living*, AAL) (STEG, STRESE, *et al.*, 2006).

Para tanto, um projeto europeu denominado AALIANCE foi criado com o propósito de incentivar a pesquisa e o desenvolvimento de soluções para AVA. Atualmente, a segunda versão do projeto, AALIANCE2, visa estabelecer uma rede sustentável de interação e padronização das iniciativas atualmente desenvolvidas por setores públicos e privadas (AALIANCE2, 2015).

Deste esforço surgiram direta ou indiretamente algumas propostas técnico-científicas em AVA, tais como: Alhambra (DIMITROV, 2005), PERSONA (SOLER, PEÑALVER, *et*

al., 2010), OASIS (OASIS-PROJECT, 2015), UniversAAL (UNIVERSAAL, 2015), OpenAAL (OPENAAL, 2015), Hydra (HYDRA, 2015) e OpenHAB (OPENHAB, 2017).

Estas iniciativas visam endereçar alguns dos desafios encontrados na área: interoperabilidade e integração de dispositivos médicos, segurança, privacidade e proteção de dados, projeto, arquitetura e metodologia de desenvolvimento, atributos de qualidade (alta disponibilidade, consistência, robustez etc), padronização, soluções abertas e experiência de uso (MEMON, WAGNER, *et al.*, 2014).

Todavia, alguns estudos denotam a carência de pesquisa e desenvolvimento de sistemas para AVA com capacidade de detectar e tolerar falhas: Chetan, Ranganathan e Campbell (2005), de Andrade Tarrinho (2009) e Rodrigues *et al.* (2012). Cabe ressaltar que o conceito de falha em AVA não difere muito do empregado em outros sistemas distribuídos haja vista que os riscos de disponibilidade a que este ambiente está exposto é o mesmo encontrado, por exemplo, em redes móveis auto organizáveis: perda de sinal de transmissão, limitações de tráfego, QoS e outras falhas de rede que podem deixar os dispositivos inacessíveis ou inoperantes (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Tem-se, por conseguinte, que a manifestação de falhas sem o devido tratamento pode comprometer sobremaneira a missão do sistema para AVA qual seja a manutenção do bem estar e a vida da pessoa assistida. Destarte, um relevante problema técnico-científico se apresenta: num país em desenvolvimento e atravessando o fenômeno da transição demográfica como desenvolver um sistema para AVA economicamente acessível e tolerante a falhas? O enfrentamento de tal problemática é o objetivo do presente estudo.

1.1. Objetivos

O objetivo principal deste trabalho é propor e desenvolver um subsistema específico para detecção de falhas em sistemas para Ambiente de Vida Assistida por meio do emprego de tecnologias abertas (*open hardware/software*).

1.1.1. Objetivos Específicos

Para alcançar o objetivo primário anterior, alguns objetivos específicos têm que ser atendidos, a saber:

a) delinear os erros e as falhas que podem comprometer o adequado funcionamento de sistema para controle de Ambiente de Vida Assistida;

b) encontrar paradigmas de detecção de erros em sistemas distribuídos e a partir destes desenvolver um módulo com o fim de prover maior dependabilidade ao sistema para Ambiente de Vida Assistida empregado neste trabalho;

c) implantar com o emprego de tecnologias abertas um sistema tolerante a falhas em protótipo de Ambiente de Vida Assistida de modo a comprovar a viabilidade da proposta;

d) por fim, avaliar a desempenho na detecção de erros quando utilizado o módulo desenvolvido pelo presente estudo em contexto de sistema para Ambiente de Vida Assistida.

1.2. Organização do Trabalho

Esta dissertação apresenta-se divida em seções, a seção 2 oferece uma revisão bibliográfica, fazendo uma breve abordagem dos conceitos e características inerentes ao Ambiente de Vida Assistida (subseção 2.1). Cita aspectos de arquitetura e de plataforma tecnológica empregada em AVA (subseção 2.2), como também, discute a necessidade de medidas de tolerância a falhas em soluções desenvolvidas para AVA (subseção 2.3) e, por fim, discute os trabalhos correlatos desenvolvidos na área (subseção 2.4). A metodologia, materiais e métodos utilizados serão apresentados na seção 3. O protótipo de AVA construído para execução do experimento encontra-se descrito na seção 4. Os resultados do presente trabalho são expostos na seção 5. Por fim, tem-se na seção 6 a conclusão e a sugestão de trabalhos futuros.

2. Revisão Bibliográfica

As subseções a seguir têm por finalidade apresentar: os conceitos e características de Ambiente de Vida Assistida (AVA) (subseção 2.1), o projeto e a arquitetura de sistema para AVA (subseção 2.2), a necessidade e como se desenvolver medidas de dependabilidade em sistemas para AVA (subseção 2.3) bem como trabalhos correlatos desenvolvidos na área (subseção 2.4).

2.1. Ambiente de Vida Assistida

Antes de se tratar de sistema para Ambiente de Vida Assistida, faz-se útil conhecer o que seja Ambiente de Vida Assistida, AVA, (*Ambient Assisted Living*, AAL). O termo AVA refere-se a ideias, produtos e serviços que visam aperfeiçoar a qualidade de vida, seja física ou mental, incluindo a independência, conforto, segurança e o bem-estar social de pessoas idosas ou portadoras de necessidades especiais. O tema AVA emergiu na década de 1990, entretanto, apenas em meados da década de 2000 passou-se a ter ações efetivas nesta área, seja por parte de governos, comunidade científica e indústria (NAKAGAWA, ANTONINO, *et al.*, 2013).

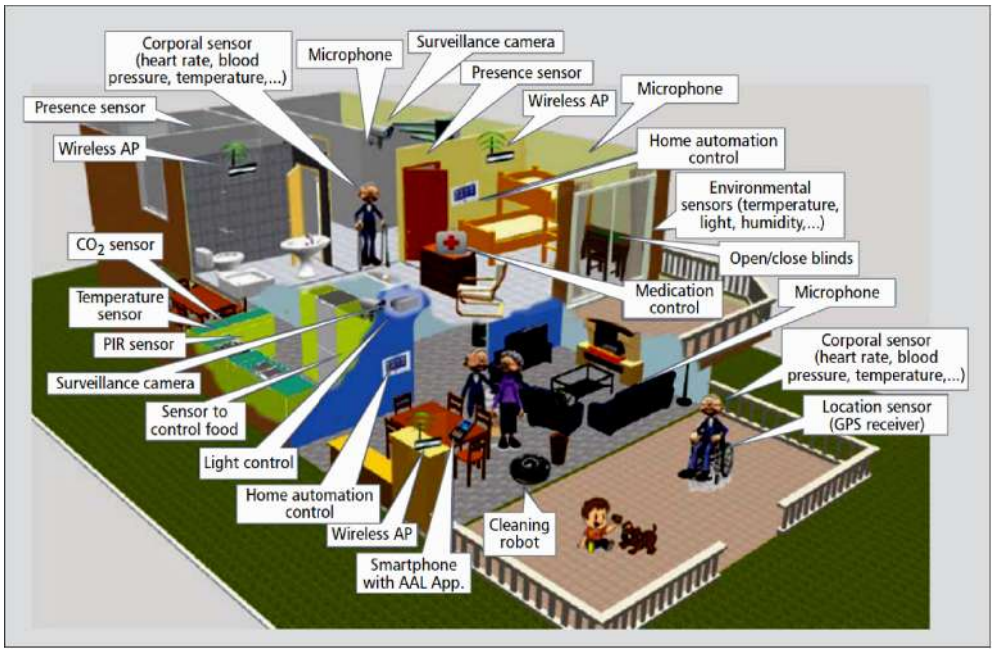
Portanto, Ambiente de Vida Assistida é, essencialmente, uma proposta científica e tecnológica que visa auxiliar a manutenção da saúde e o bem-estar de pessoas. Para tanto, utiliza-se de dispositivos inteligentes[1]: braceletes, roupas, camas, quartos etc., que visam ampliar a autonomia dos indivíduos na execução das mais simples atividades diárias, bem como prover remotamente serviços de saúde e de assistência social (SUN, DE FLORIO, *et al.*, 2010).

Almeja-se que em um AVA existam facilidades adaptáveis ao perfil e ao contexto do indivíduo assistido, configuradas por meio da interação de uma miríade de dispositivos inteligentes dispostos nos mais diversos ambientes em que uma pessoa possa se encontrar, sempre considerando que as pessoas assistidas podem ser dotadas de limitações, sejam: deficiências visuais, auditivas, restrições de mobilidade ou falta de destreza (PIEPER, ANTONA e CORTÉS, 2011).

[1] Dispositivo inteligente (embarcado) em AVA são basicamente sensores e atuadores dirigidos por microcontrolador interconectado em rede com um ou mais computadores que ou assumem o papel de ponte para uma rede externa (gateway) ou atuam como controladores autônomos da rede local, comumente denominado espaço AVA.

A Figura 1 ilustra uma representação de Ambiente de Vida Assistida:

Figura 1 : Representação de um AVA (LLORET, CANOVAS, *et al.*, 2015)



Tendo em vista que o principal objetivo de um AVA seja prover serviços que auxiliem na manutenção da saúde e no bem-estar de pessoas dotadas de limitações físicas, faz-se natural exigir o emprego de tecnologias confiáveis e economicamente acessíveis. Do contrário, poder-se-á na eventualidade de uma simples falha, seja de projeto ou mesmo operação, comprometer a manutenção da saúde e, consequentemente, a vida de pessoas assistidas. Neste sentido, um AVA pode ser considerado um sistema de missão crítica (RODRIGUES, ALVES, *et al.*, 2012) e (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Todavia, antes de se analisar a necessidade do cumprimento de requisitos de dependabilidade, através da adoção medidas de tolerância a falhas, em sistema para AVA, faz-se necessário conhecer a estrutura dessa proposta tecnológica.

2.2. Sistema para Ambiente de Vida Assistida

Conforme enunciado, sistemas para AVA visam cumprir a missão de assistir pessoas com necessidades especiais. Pode-se, portanto, deduzir que os serviços de cuidado ofertados à pessoa assistida ocorrerão de duas maneiras: em resposta à iniciativa das pessoas ou automaticamente.

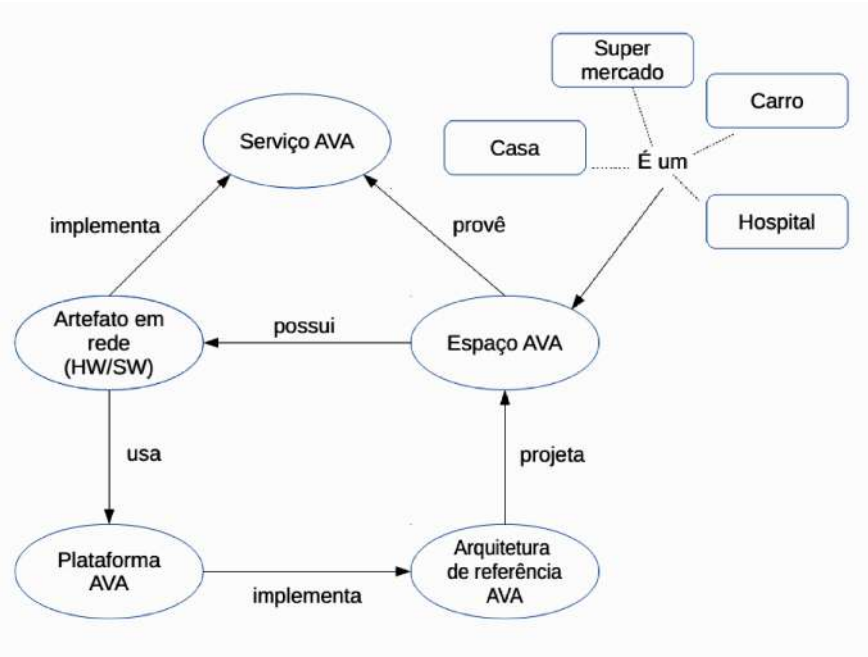
Entretanto, a assistência automática em AVA exige a existência de mecanismos que

possibilitem a percepção e a sensibilidade de avaliar o contexto do meio bem como a capacidade preditiva acerca do estado clínico e do comportamento do indivíduo, assim como de seu ambiente (RODRIGUES, ALVES, *et al.*, 2012).

Sistemas em AVA, portanto, oferecem serviços de assistência médica e pessoal (AAL *Services*) por meio da integração de dispositivos inteligentes em rede (*Networked Artifacts* SW/HW) alocados em espaços AVA (AAL *Spaces*): casa, condomínio, shoppping center, carro, hospital etc. Para suportar *Networked Artifacts* em AAL *Spaces* há um componente responsável pela integração e abstração da heterogeneidade existente entre os espaços e os dispositivos interconectados, a plataforma AVA (AAL *Platforms*).

Em termos conceituais AAL *Platform* materializa a arquitetura de referência AVA (AAL *Reference Architeture*) que é, tão somente, o paradigma estrutural de um sistema para AVA. A Figura 2 faz uma representação do relacionamento existente entre os principais elementos descritos que compõe um AVA (TAZARI, FURFARI, *et al.*, 2012).

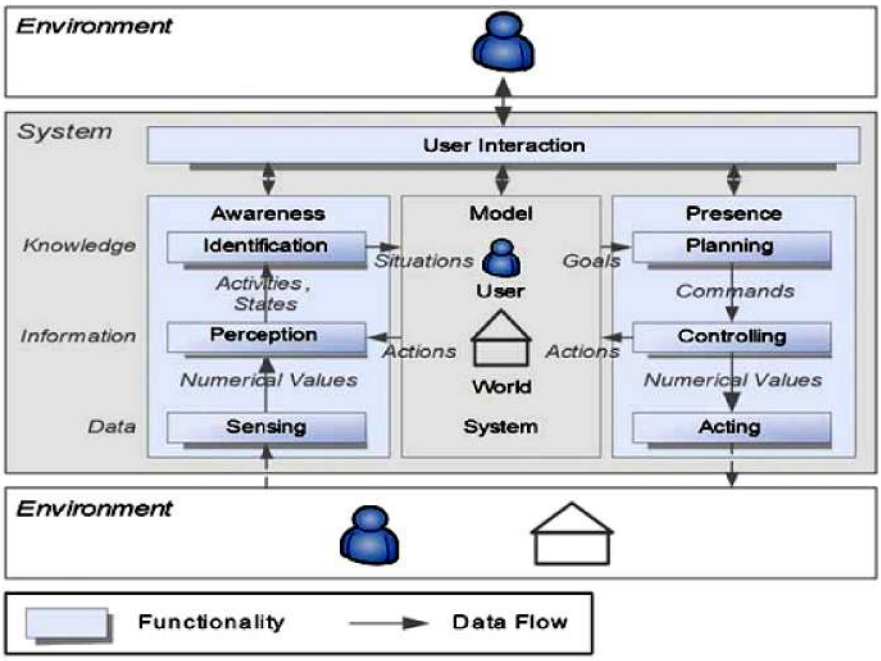
Figura 2 : Modelo conceitual de um sistema para AVA (TAZARI, FURFARI, *et al.*, 2012)



A arquitetura de referência de Nehmer *et al.* (2006) estipula a existência de dois módulos principais para sistemas em AVA: Contexto e Presença (Figura 3). O módulo *Contexto* é responsável pela compreensão da situação clínica do indivíduo, seja no aspecto físico ou mental, como também das variáveis espaciais em que está inserido. *Presença*, por sua vez, gerencia as funcionalidades que diretamente interferem no estado da pessoa assistida e do ambiente, como, por exemplo, alertas, comandos ou ações produzidas pelo

sistema.

Figura 3 : Arquitetura de referência em Ambiente de Vida Assistida (NEHMER, BECKER, *et al.*, 2006)



Para satisfazer as atribuições de cada módulo, há a necessidade de se decompô-los em elementos internos. *Contexto* pode ser subdivido nas seguintes camadas: a) sensores responsáveis pela coleta informações físicas e, posteriormente, pela tradução dessas em dados numéricos; b) percepção transforma dados numéricos em representações simbólicas através da fusão de dados sensoriais ou da avaliação de estados em fluxos de estado de máquina, tais como, avaliação da temperatura corporal, frequência cardíaca etc.; c) identificação avalia em maior nível de abstração os diversos padrões de comportamento e, a partir disso, infere desvios de conduta em busca de situações de alerta ou emergência (RODRIGUES, ALVES, *et al.*, 2012).

O módulo de Presença, por conseguinte, possuirá os seguintes componentes: a) planejamento avalia a situação atual por meio dos dados recebidos e compara com os objetivos do sistema para decidir se o momento presente exige uma reação e de que tipo; b) controlador: executa ações, decompondo inicialmente as ações planejadas em tarefas menores para, em seguida, traduzi-las em dados numéricos; c) atuador influencia no meio e no estado do indivíduo através da coleta dos valores enviados pela camada anterior (RODRIGUES, ALVES, *et al.*, 2012).

Compreendido o conceito, características e arquitetura de um sistema para AVA faz-se

necessário discutir a necessidade do cumprimento de requisitos de dependabilidade do sistema, especialmente através da adoção de medidas de tolerância a falhas.

2.3. Dependabilidade em sistema para Ambiente de Vida Assistida

Conforme mencionado, uma das principais características de um sistema para AVA é a capacidade de estar apto e sensível ao contexto. Dessa forma, a inferência correta do estado clínico e da situação do meio é primordial para garantir a manutenção do bem-estar e a vida do indivíduo (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Como assegurar, neste contexto, que o sistema não esteja interpretando uma situação equivocada da realidade? Ou ainda, como prevenir repostas inadequadas ou omissões irremediáveis? Apenas com o cumprimento eficiente dos aspectos delineados em dependabilidade de sistemas (*software dependability*), como disponibilidade, confiabilidade, segurança e manutenabilidade (capacidade de manutenção) (TANENBAUM e VAN STEEN, 2007) poder-se-á minimizar tais circunstâncias inaceitáveis para um sistema em Ambiente de Vida Assistida.

Em AVA temos a possibilidade de uso de uma miríade de dispositivos: PCs, *notebooks*, *smartphones*, sensores, atuadores, telas, unidades de áudio, *scanners*, câmeras e projetores. Portanto, há um cenário bastante heterogêneo e, por isso, propício a falhas inerentes à falta de padronização das tecnologias empregadas e que, por isso, precisam ser analisadas e tratadas de forma adequada. Por exemplo, em dispositivos móveis: *notebooks*, *smartphones* e sensores, há limitações físicas: limite de cobertura de comunicação, autonomia energética e resistência mecânica que podem contribuir para o surgimento de falhas e, consequentemente, operação incorreta do sistema AVA (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Um sistema completamente tolerante a falhas, caso isso seja possível, necessita do cumprimento de um conjunto elevado de requisitos não funcionais. Portanto, faz-se necessário levantar tais pressupostos de qualidade do sistema para, em seguida, efetivar medidas de tratamento a falhas em tecnologias de hardware (dispositivos), rede, aplicação e serviços (CHETAN, RANGANATHAN e CAMPBELL, 2005).

No domínio de rede em AVA tem-se, como exemplo, a elevada probabilidade de perda do sinal de transmissão, limitações de tráfego, QoS, e outras falhas de rede que podem deixar dispositivos móveis inacessíveis, ocasionando a interpretação incorreta das

informações compartilhadas. Por isso, torna-se urgente a adoção de medidas automáticas de detecção e tratamento de falhas de modo a prevenir e mitigar tais situações indesejadas (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Em contrapartida, ao se cogitar a adoção de medidas de prevenção a falhas há de se ater também ao processo de desenvolvimento de software, com a inclusão de atividades de depuração, testes e verificação, tradicionalmente desconsideradas por razões de ordem orçamentária, haja vista que a adição de tais atividades comprometerem de 50 a 75% do custo total do projeto (HAILPERN e SANTHANAM, 2002).

Mesmo assim, sabe-se que *bugs* de vários níveis de severidade remanescem em softwares rigorosamente testados (WHITTAKER, 2000). Não obstante, é de conhecimento geral que falhas em softwares podem derivar de erros em sistemas operacionais, exceções não capturadas, falhas na utilização, vírus e interferência de outros softwares maliciosos entre outros motivos não esperados (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Diante desse cenário, vislumbra-se uma grande oportunidade de pesquisa e desenvolvimento de técnicas automáticas de detecção e tratamento de falhas em sistema para Ambiente de Vida Assistida. Mesmo sabendo que a temática de tolerância a falhas já possui algumas décadas de estudo, sejam em disciplinas de arquitetura de sistemas, sistemas operacionais, sistemas distribuídos, rede de computadores e sistemas móveis, faz-se necessário que em cada nova área de aplicação da tecnologia esse tema seja reapreciado, pois no contexto de sistema para AVA emerge um conjunto próprio de questões para os quais as soluções anteriormente projetadas podem ter aplicabilidade limitada ou ineficiente (CHETAN, RANGANATHAN e CAMPBELL, 2005).

Por conseguinte, faz-se necessário levantar o cenário do cumprimento dos requisitos de dependabilidade em sistema para Ambiente de Vida Assistida, especificamente em plataformas AVA, de modo a se propor, desenvolver e avaliar o cumprimento de medidas que propiciem o robusto e correto funcionamento do sistema.

2.3.1. Requisitos de dependabilidade em sistema para Ambiente de Vida Assistida

Como visto, para garantir o funcionamento pleno de um sistema para AVA tem-se por necessário prover além das facilidades desejadas o cumprimento dos requisitos não funcionais, tais como disponibilidade e segurança de uso. Neste sentido, alguns sistemas em AVA, além de tratar dos requisitos de negócio, desenvolveram também os de ordem qualitativa. Cabe ressaltar, entretanto, que entre as soluções avaliadas houve consideráveis

diferenças quanto ao cumprimento dos aspectos não funcionais do sistema. Por exemplo, o uso de medidas de tolerância a falhas não foi levado em consideração por muitas delas mesmo sabendo que a adoção de tais medidas mitigaria a ocorrência de riscos (NAKAGAWA, ANTONINO, *et al.*, 2013).

Não obstante, nas plataformas AVA baseadas em tecnologia OSGi (OSGI, 2017), considerada sob os critérios da modularidade, manutenabilidade e eficiência como a infraestrutura mais apropriada para desenvolvimento de *middleware* em sistema para AVA, observou-se o pelo menos o tratamento nativo do ambiente de execução (OSGi *runtime*) aos requisitos de dependabilidade (ANTONINO, SCHNEIDER, *et al.*, 2011).

Das plataformas AVA construídas sob a tecnologia OSGi, tais como, Alhambra (DIMITROV, 2005), PERSONA (SOLER, PEÑALVER, *et al.*, 2010), OASIS (OASIS-PROJECT, 2015), UniversAAL (UNIVERSAAL, 2015), OpenAAL (OPENAAL, 2015), Hydra (HYDRA, 2015) e OpenHAB (OPENHAB, 2017), avaliou-se o cumprimento de medidas de tolerâncias a falhas tais como autorrecuperação[2], eliminação de ponto único de falha[3], segurança de uso (*safety*)[4] e implementação de módulos de redundância (ANTONINO, SCHNEIDER, *et al.*, 2011).

Para Antonino, Schneider *et al.* (2011) quanto ao aspecto autorrecuperação, todas possuem, no mínimo, o mecanismo provido pelo ambiente de execução OSGi. Ademais, (i) Hydra tem um mecanismo específico para detectar falhas no sistema; (ii) PERSONA e UniversAAL implementaram medidas de autorrecuperação também em nível de *hardware*; (iii) Alhambra, OASIS, OpenHAB e OpenAAL não forneceram mecanismos adicionais para recuperação da plataforma na ocorrência de falhas.

Para os autores supracitados quanto à eliminação de ponto único de falha observou-se que: (i) Hydra por ter todas as comunicações realizadas pelo serviço *Network Manager* possui em seu único servidor denominado *Super Node* a existência de um ponto único de falha não remediado; (ii) PERSONA foi projetada e desenvolvida para se evitar ponto único de falhas. No entanto, há ainda componentes importantes no topo da arquitetura que tornam o nó hospedeiro um ponto crítico de falha, por exemplo, se o nó que hospeda o serviço *Dialog Manager* falhar há a perda de comunicação do sistema com o usuário; (iii) Em

[2] Autorrecuperação genericamente pode ser entendida como a habilidade do sistema recuperar-se a si mesmo na ocorrência de uma falha (ANTONINO, SCHNEIDER, *et al.*, 2011).
[3] Eliminação de ponto único de falha é a medida que visa afastar a vulnerabilidade existente quando um único componente possui a capacidade de comprometer o adequado funcionamento de todo o sistema (ANTONINO, SCHNEIDER, *et al.*, 2011).
[4] Segurança de uso pode ser compreendida como a satisfação constante da eficácia do sistema obtida em razão de medidas arquitetônicas adotadas que visam sempre garantir a preservação do estado correto do sistema (ANTONINO, SCHNEIDER, *et al.*, 2011).

UniversALL, por sua vez, não se encontrou na documentação disponível algo que pudesse ser tido como ponto único de falha; (iv) Alhambra, OpenHAB assim como Hydra, possuem todas um *gateway* residencial que torna este elemento um ponto único de falha da plataforma inteira; (v) OASIS por ter apenas um componente para registro de serviços AVA pode-se considerá-lo como ponto único de falha para a tarefa de registro e invocação de serviços existentes e, por fim, em (vi) OpenALL não se encontrou qualquer informação a respeito de tratamento de ponto único de falha.

Quanto à implementação específica de medidas de redundância tem-se que em (i) Hydra houve o emprego de diferentes serviços para o cumprimento de um mesmo objetivo; (ii) UniversAAL, por seu turno, visou implementar mecanismos de suporte a instalação redundante de componentes críticos. Nas demais soluções analisadas não se encontraram medidas de redundância (ANTONINO, SCHNEIDER, *et al.*, 2011).

Em resumo, da análise efetuada por Antonino, Schneider *et al.* (2011) sobre as plataformas AVA quanto ao cumprimento aos requisitos de dependabilidade, especialmente referente à adoção de providências relativas à tolerância a falhas, estabeleceu-se na Tabela 1 uma escala de verificação sobre as plataformas AVA, onde os indicadores:

- Amplamente Abordado (AA) sinaliza que medidas de tolerância a falhas foram amplamente abordadas pela proposta em questão;
- Abordado (A) informa que aspectos de tolerância a falhas foram abordados pela solução em análise, mas de forma simples;
- Não Abordado (NA) indica que medidas de tolerância a falhas não foi em nenhum aspecto abordado pela proposta em verificação, e;
- Não Localizado (NL) denota que nenhuma informação a respeito de providências relativas à tolerância a falhas foi localizada na documentação disponível da solução.

Tabela 1 : Avaliação de requisitos de tolerância a falhas em sistemas para AVA (ANTONINO, SCHNEIDER, *et al.*, 2011).

Solução	Requisitos de confiabilidade		
	Autorrecuperação	Eliminação de ponto único de falha	Medidas de redundância
UniversAAL	AA	AA	A
Hydra	AA	NA	A
PERSONA	AA	NA	NA
OASIS	NA	NA	NA
Alhambra	NA	NA	NA
OpenHAB[5]	NA	NA	NA
OpenAAL	NA	NL	NA

Percebe-se, portanto, que em nenhuma plataforma encontrou-se, em plenitude, os requisitos de tolerância a falhas avaliados. Entretanto, vale registrar que das soluções analisadas, UniversAAL foi a mais bem posicionada. Todavia, em uma pesquisa mais minuciosa, considerando outras fontes, observou-se que a plataforma UniversAAL, embora seja uma iniciativa *opensource* importante, possui apenas quatro desenvolvedores cuja atividade mais recente data do final de 2015 (GITHUB, 2017). Dessa forma, não obstante ela aparenta ser a plataforma mais robusta em termos de especificação e projeto, observa-se a descontinuidade prática desta solução. Por isso, fez-se necessária a busca de outra proposta que também superasse esta limitação, descontinuidade.

Das outras plataformas mais bem posicionadas percebeu-se que PERSONA apenas elaborou proposições de projeto que foram aproveitadas por outras plataformas AVA. Quanto a Hydra sabe-se que o projeto patrocinador encerrou as atividades em dezembro de 2010. Desta plataforma, todavia, derivou-se em 2014 o *middleware* LinkSmart. Contudo, em razão da baixa quantidade de desenvolvedores, não há atualização dessa plataforma AVA há mais de um ano.

Das plataformas abertas restantes verificou-se que OpenHAB é a que mantém, até a presente data, iniciativas de expansão e encontra-se em pleno desenvolvimento (GITHUB, 2017). Ademais, a plataforma OpenHAB possui o suporte institucional de uma entidade filantrópica denominada OpenHAB *Foundation* (OPENHAB-FOUNDATION, 2017). Consequentemente, neste trabalho foi definida a plataforma AVA OpenHAB como sendo a ideal para elaboração de um módulo detector de falhas, tendo em vista, a necessidade de se implementar medidas de tolerância a falhas neste sistema vital para as pessoas assistidas.

Dessa forma, faz-se necessária a análise minuciosa dos componentes estruturais desta plataforma AVA de modo a identificar possíveis pontos de melhoria no que se refere ao

[5] A plataforma OpenHAB não se encontra presente no estudo comparativo de (ANTONINO, SCHNEIDER, *et al.*, 2011), todavia foi analisada pelo presente estudo e, por isso, incorporada a esta seção.

cumprimento de requisitos de tolerância a falhas.

2.3.2. OpenHAB, plataforma para sistema tolerante a falhas em AVA

Conforme citado anteriormente, o elemento do sistema para AVA responsável pela integração e abstração da heterogeneidade dos dispositivos interconectados é a plataforma AVA (AAL *Platforms*), que implementa a arquitetura de referência AVA (*AAL Reference Architeture*). Diante disso, torna-se necessário conhecer em detalhes a estrutura da plataforma AVA escolhida, OpenHAB. Cabe destacar que a plataforma AVA escolhida fará parte das variáveis independentes do experimento executado neste trabalho (seção 3).

OpenHAB, *open Home Automation Bus* (OPENHAB, 2017), é uma das mais populares soluções *opensource* para implantação de ambientes inteligentes. A plataforma visa prover o controle total bem como a flexibilidade máxima do espaço físico para o usuário (OPENHAB-FOUNDATION, 2017).

Existem outras soluções de domótica e Internet das Coisas, além de *gadgets* no mercado, que são úteis por si próprios. Entretanto, essas soluções são em grande parte proprietárias e, além disso, possuem ajustes e integração limitados aos demais dispositivos licenciados, por isso, mostram-se adequados somente para os casos de uso pretendidos. Não obstante, o maior problema desses sistemas e dispositivos é que a aplicação vem previamente estabelecida pelo fabricante da solução.

No entanto, acontece que o usuário e o ambiente alteram rapidamente o próprio estado suscitando mudanças no sistema que não serão suportadas pela configuração inicial do sistema. Ou ainda, tais requisitos excepcionais geralmente exigem a interação com outros sistemas de diferentes fabricantes.

Desse cenário é que OpenHAB surgiu para preencher a patente lacuna: colocar o usuário final no foco da solução e permitir que ele altere o sistema de acordo com a sua necessidade. Dessa forma, OpenHAB serve como um ponto de integração das necessidades personalizadas de automação residencial e, além disso, permite que equipamentos e tecnologias originalmente diferentes possam interagir uns com os outros independentemente de acordo entre fornecedores ou de protocolo empregado (OPENHAB, 2017). Ademais, OpenHAB possui o que é mais relevante para uma solução *opensource*, encontra-se ativo e possui uma enorme comunidade de usuários e desenvolvedores interagindo e aperfeiçoando a solução (OPENHAB-FOUNDATION, 2017).

Para prover uma integração neutra de tecnologia, protocolo e fornecedores OpenHAB foi concebido para ter uma arquitetura capaz de estabelecer regras de automação abrangentes e interfaces de usuário uniformes, inclusive provendo uma API bem definida e aberta para integração com outros sistemas (OPENHAB, 2017).

A plataforma OpenHAB baseia-se na integração local, seguindo o princípio de "Intranet das Coisas" (OPENHAB-FOUNDATION, 2017). Para melhor compreensão, seguem os componentes da estrutura do OpenHAB:

a) OpenHAB *runtime*, servidor que executa os processos e as regras definidas assim como manipula os dados coletados; e

b) OpenHAB *designer* que pode ser considerada a ferramenta de configuração da primeira.

O OpenHAB *runtime* é um conjunto de pacotes implantados em um *framework* OSGi (EQUINOX, 2017). Portanto, é uma solução Java pura e, por isso, precisa de apenas uma máquina virtual Java, JVM, para ser executada. Por ser baseado em OSGi, o *runtime* possui uma arquitetura altamente modular que possibilita adicionar e remover funcionalidades durante o tempo de execução do serviço sem a necessidade de interrompê-lo. A comunicação entre os componentes ocorre através de dois canais:

a) barramento de eventos assíncrono (OpenHAB *Event Bus*); e

b) o repositório de estados de itens (OpenHAB *Repository*).

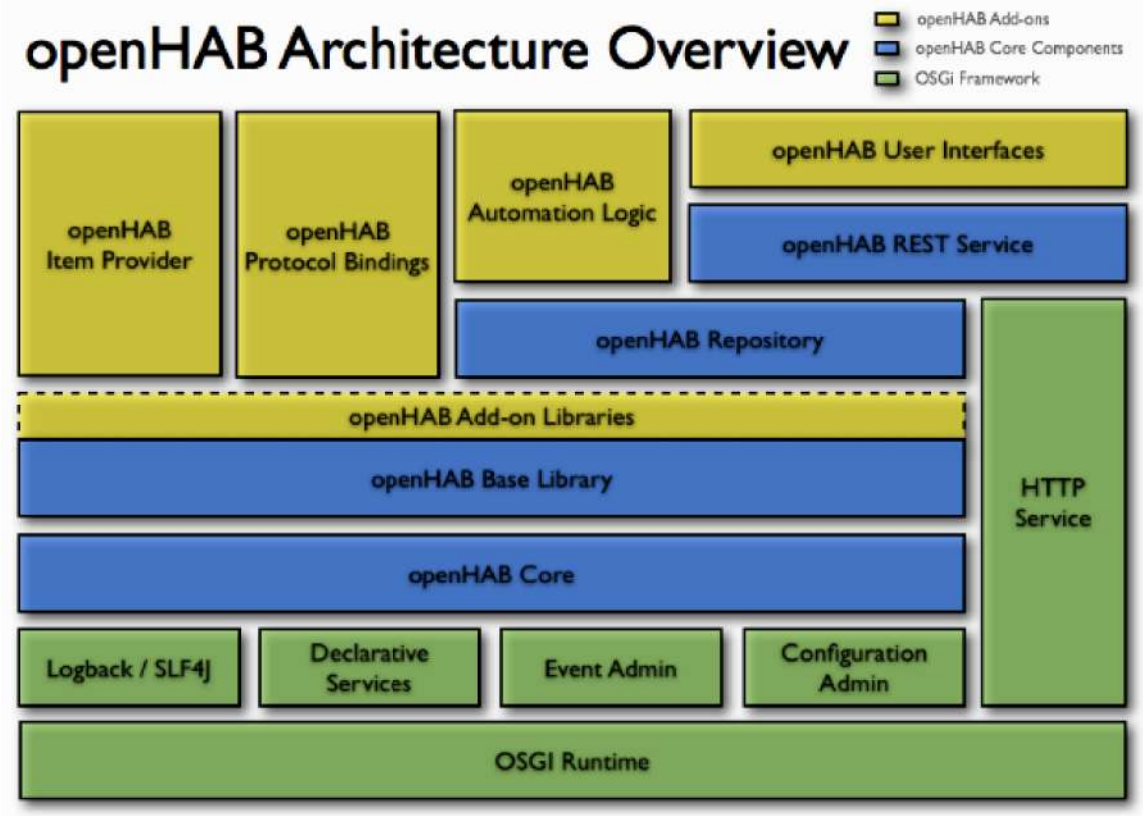
O barramento de eventos assíncrono é o serviço base do OpenHAB. Todos os *bundles* que não possuem necessidade de preservação de estado devem utilizá-lo para informar os outros *bundles* a respeito de eventos como também para serem atualizados a respeito de eventos externos. Existem principalmente dois tipos de eventos: comandos que acionam uma ação ou alguma alteração de estado de alguns itens (dispositivos) geralmente em resposta a algum comando submetido (OPENHAB, 2017).

Todas as ligações de protocolo (*bindings*) responsáveis pela integração com os dispositivos inteligentes diversos são registrados no OpenHAB *Event Bus*. Isto possibilita o baixo acoplamento da plataforma com os demais dispositivos comumente empregados no mercado de domótica, o que possibilita a natureza dinâmica pretendida pela plataforma

OpenHAB. E o serviço OSGi *EventAdmin* trata-se de uma implementação leve e pronta para uso de um *middleware* do tipo *publisher-subscriber*, que atende perfeitamente aos requisitos exigidos pela solução (OPENHAB, 2017).

Na Figura 4 vê-se uma visão geral dos principais componentes (*bundles*) do OpenHAB *runtime* e como esses elementos se relacionam (OPENHAB, 2017):

Figura 4 : Arquitetura do OpenHAB (OPENHAB, 2017).



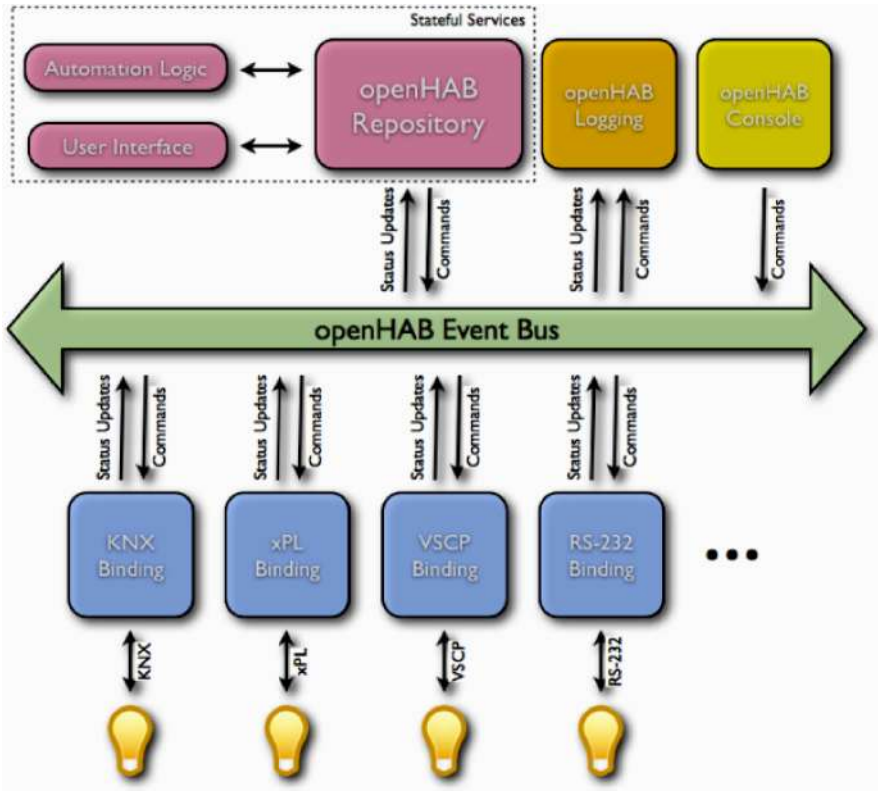
Cabe ressaltar, todavia, que originalmente o OpenHAB não foi concebido originalmente para hospedar-se em dispositivos embarcados ou inteligentes, por isso, não possui em sua estrutura um elemento responsável pela comunicação remota com outras instâncias OpenHAB distribuídas. Ao invés disso, OpenHAB visa estabelecer um barramento de integração aos diferentes protocolos que são utilizados por esses dispositivos.

Em uma instalação típica haverá geralmente apenas uma instância do OpenHAB em execução em algum computador central do ambiente (*gateway*). Entretanto, como o serviço OSGi *EventAdmin* também pode ser usado como um serviço remoto, é possível conectar várias instâncias OpenHAB distribuídas através do OpenHAB *Event Bus* (OPENHAB, 2017).

Ademais, sabe-se que nem toda funcionalidade pretendida pelo usuário pode ser satisfeita simplesmente através de serviços sem estado. Por esta razão, o OpenHAB possui o "Repositório de Itens" (OpenHAB *Repository*), que está conectado ao OpenHAB *Event Bus* e mantém o controle do estado atual de todos os itens (dispositivos) do sistema. Neste sentido, o mecanismo de execução da lógica de automação (OpenHAB *Automation Logic*) sempre o utiliza para consulta de modo a manter-se informado dos estados atuais dos dispositivos.

Em acréscimo, OpenHAB *Repository* possibilita a salvaguarda dos estados dos componentes, como também garante que o estado encontra-se em sincronia para todos *bundles* e, por fim, viabiliza a persistência dos estados para um sistema de arquivos ou banco de dados, a fim de ser mantidos e recuperados em um eventual reinício do sistema (OPENHAB, 2017). Na Figura 5 tem-se a representação das comunicações internas estabelecidas nesses canais.

Figura 5 : Comunicação interna do OpenHAB (OPENHAB, 2017).



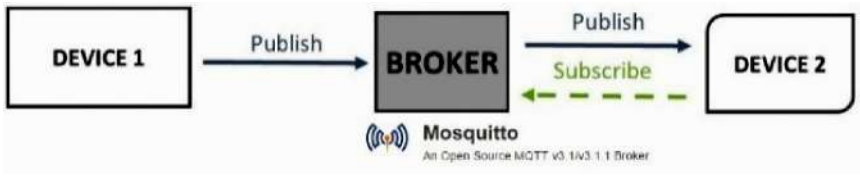
Como destacado anteriormente, verifica-se que o OpenHAB apesar de ser apropriado para se implementar uma plataforma AVA confiável não recomenda a integração nativa de múltiplas instâncias da solução, descumprindo um requisito imprescindível para eliminação

de ponto único de falha. Por isso, consultando fóruns da comunidade de usuários do OpenHAB buscou-se uma alternativa segura e prática para integrar múltiplas instâncias da plataforma AVA. A melhor proposta encontrada sugere a vinculação de cada instância OpenHAB com um servidor MQTT pareado que suporte a funcionalidade *bridge*, ou seja, a propagação de canais entre servidores MQTT (OPENHAB-COMMUNITY, 2017).

Para maior compreensão, tem-se que MQTT significa *Message Queue Telemetry Transport* (Enfileiramento de mensagens de telemetria transportada). Esta tecnologia de colocação de mensagens em fila permite que aplicações em execução em momentos diferentes (assíncronas) se comuniquem através de redes heterogêneas mesmo que tais sistemas possam estar temporariamente sem comunicação (*offline*).

As aplicações (tal como OpenHAB) enviam, recebem ou consultam (leem sem remover) as mensagens das filas (MICROSOFT-DEVELOPER-NETWORK, 2017). MQTT, portanto, é um protocolo de mensagens do tipo *publisher-subscriber*, extremamente simples e leve, projetado para dispositivos com poucos recursos e dotados de baixa largura de banda, alta latência ou instalados em redes não confiáveis (MQTT, 2017).

Figura 6 : Esquema de funcionamento do broker publisher-subscriber Mosquitto (SANTOS, 2017).



Os princípios de projeto do protocolo MQTT são, portanto, minimizar a largura de banda consumida e aperfeiçoar o uso de recursos do dispositivo enquanto também visa prover confiabilidade em algum grau de garantia de entrega. Esses princípios o tornam protocolo ideal do mundo emergente "máquina-a-máquina" (M2M) ou "Internet das coisas". Ou seja, o protocolo MQTT é voltado para dispositivos embarcados interconectados e para aplicações móveis onde largura de banda e energia de bateria são recursos escassos (MQTT, 2017).

Figura 7 : Publicação de dados sensoriais via MQTT (SANTOS, 2017).

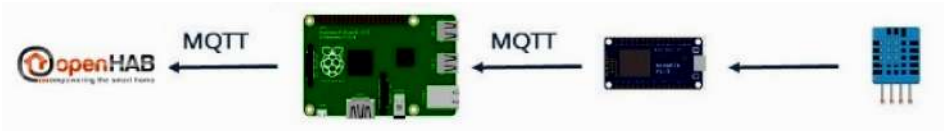
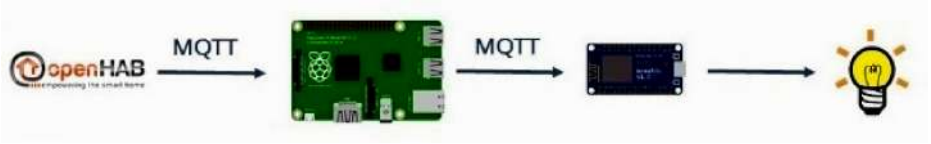


Figura 8 : Subscrição de comandos OpenHAB via MQTT (SANTOS, 2017).



MQTT foi inventado pelo Dr. Andy Stanford-Clark da IBM e Arlen Nipper da Arcom (atual Eurotech), pelos idos de 1999. A partir de então vem sendo desenvolvido pela indústria em diversos projetos, especialmente da empresa Eurotech. Em novembro de 2011, a IBM e a Eurotech anunciaram sua participação conjunta no consórcio Eclipse M2M *Industry Working Group* e materializaram a doação do código MQTT para os cuidados do projeto Eclipse Paho.

A especificação do protocolo MQTT foi por muitos anos apresentada publicamente com uma licença *royalty-free* (MQTT, 2017). Desde julho de 2016, MQTT tornou-se um padrão internacional ISO/IEC 20922 por meio da ação do consórcio OASIS. Dessa forma, acredita-se que ainda haverá uma expansão ainda maior de uso desta tecnologia no universo da Internet das Coisas (OASIS-CONSORTIUM, 2017).

No setor de saúde, os profissionais usam o protocolo MQTT para interagir com dispositivos biomédicos, como monitores de pressão arterial e de níveis de oxigênio no sangue. De outro lado, empresas de petróleo e gás utilizam MQTT para monitorar milhares de quilômetros de dutos de óleo e derivados. MQTT, portanto, é um facilitador fundamental para a telemática, *infotainment*[6] e outras aplicações de veículos conectados. MQTT também vem se tornando cada vez mais popular para aplicações móveis interativas (OASIS-CONSORTIUM, 2017). Ademais, existem variantes do protocolo que visam estender o uso do MQTT para além do TCP/IP é o caso da proposta MQTT-SN (MQTT, 2017).

Em termos de segurança da informação pode-se utilizar, a partir da versão 3.1 do protocolo, o recurso de controle de acesso por pacote MQTT. A criptografia em toda a rede pode ser tratada com SSL (vale a destacar que SSL não é o mais leve dos protocolos, e adiciona uma sobrecarga de rede significativa). Por isso, segurança adicional pode ser proporcionada por um aplicativo de criptografia que enviam e recebem dados, mas isso não é algo embutido no protocolo MQTT, pois o propósito principal é mantê-lo simples e leve

[6] *Infotainment* é informação baseada em conteúdo de informação que também inclui conteúdo de entretenimento e que na vertente que, por exemplo, pode ser adicionado a veículos, a fim de melhorar a experiência de ser condutor ou passageiro (PAULO, 2017).

(MQTT, 2017).

Em termos de implementação do protocolo MQTT selecionou-se para este estudo a solução *opensource* Eclipse Mosquitto, tendo em vista ser uma versão extremamente leve e aplicada em redes amplas (de até 100.000 nós) com um consumo mínimo de recursos dos dispositivos e da banda de comunicação. Ademais, esta versão de servidor MQTT possui a funcionalidade *bridge* que possibilita a integração entre diversas instâncias Mosquitto (MOSQUITTO, 2017), por meio da propagação de canais, e consequentemente elimina qualquer ponto único de falha ao estabelecer redundância da plataforma AVA adotada.

Na sequência, levantar-se-á propostas adequadas para implementação de módulo de detecção e tolerância a falhas na plataforma AVA escolhida, OpenHAB mais Mosquitto.

2.3.3. Tolerância a falhas em sistemas para Ambiente de Vida Assistida

Toda atividade de tolerância a falhas inicia a partir da fase de detecção do erro e não da falha, propriamente. Dessa forma, importa compreender o sentido específico dos termos: falha, erro e defeito empregados pela área de confiabilidade de sistemas (*software dependability*).

Segundo Jalote (1994), partindo da definição de sistema oferecida por Anderson e Lee (1981) e Robinson (1982) e dada uma especificação *A* para um sistema *S* qualquer, uma falha *F* em *S* ocorre quando o comportamento *C* de *S* desvia do requerido por *A*. Em síntese, o sistema apresenta uma falha quando não consegue prover o serviço desejado pela especificação (ANDERSON e LEE, 1981). Entende-se por entrega de serviço de um sistema o comportamento percebido por outro(s) sistema(s) especial(is) que interage(m) com o mesmo, por exemplo, seu(s) usuário(s) (LAPRIE, 1985).

Já um erro é aquela parte do estado do sistema que pode ocasionar a falha (LAPRIE, 1985). Se há um erro no estado de um sistema, então poderá existir uma sequência de ações que podem ser executadas pelo sistema e que provocará uma falha, a menos que alguma medida corretiva seja anteriormente empregada. Por fim, a causa ou origem de um erro é um defeito (JALOTE, 1994).

A existência de um defeito proporciona a ocorrência de um erro inicialmente latente que pode se tornar efetivo quando de repente ativado pelo sistema ou meio. Quando o erro se torna ativo há comprometimento da entrega do serviço do sistema e, portanto, há a manifestação de uma falha. Em resumo, um erro é manifestação exterior de um defeito e,

consequentemente, a falha é a manifestação percebida de um erro existente no serviço prestado pelo sistema (LAPRIE, 1985).

Para efeitos de simplificação adotar-se-á, neste trabalho, o modelo de falha por parada (*crash*), um modo benigno (seguro) de falha em que o processo simplesmente cessa de responder as requisições a ele submetidas, haja vista ser este o tipo de erro mais comumente tratado pela literatura específica: Tai, Tso e Sanders (2004), Friedman e Tcharny (2005), Sridhar (2006), Greve, Sens, *et al.* (2011) e Greve, Sens, *et al.* (2012).

Tendo o conceito de serviço como uma abstração comportamental do sistema, percebe-se, portanto, a forte dependência do conceito falha com a real aplicação do sistema em questão (LAPRIE, 1985). Diante disso, o presente estudo vislumbra que os processos distribuídos que compõe um sistema para AVA precisam ser monitorados através de um módulo que verifique constantemente se os mesmos se encontram ativos e disponíveis para a satisfação dos requisitos desejados: manutenção da saúde e bem-estar do indivíduo. Todavia, desconsiderar-se-á neste trabalho os aspectos referentes ao consumo de energia do módulo detector de erros, podendo, dessa forma, ser examinado em estudos posteriores.

2.3.3.1. Detecção de erros em sistemas distribuídos

Como visto anteriormente, um erro é parte do estado do sistema que pode e deve ser detectada. Por isso, um componente do sistema deve realizar continuamente rotinas de verificação de modo a inferir precocemente a manifestação de uma falha. A este componente dá-se o nome de "detector de erro" ou "detector de falha". Ademais, a efetividade do módulo detector de erros ou falhas garante a eficácia de um mecanismo de tolerância a falhas (JALOTE, 1994).

Assumindo que o uso de redundância de nós fornece o mascaramento e a resiliência de processamento em sistemas distribuídos, antes se deve ter por premissa que os membros não faltosos deverão ser capazes de decidir quem ainda é um membro e/ou quem, de repente, deixou de sê-lo, ou seja, possuem a capacidade de detectar quando um membro falhou ou deixou o sistema (TANENBAUM e VAN STEEN, 2007)

Tradicionalmente há duas abordagens para se detectar falhas de processos distribuídos. Na primeira os processos enviam ativamente uns aos outras mensagens do tipo: "você está vivo?" (para as quais obviamente aguardam-se respostas), ou ainda, aguardam passivamente pela entrada mensagens de processos diferentes de si. A segunda abordagem, apenas faz sentido quando se puder garantir que há comunicação suficiente

entre os processos. A prática costumeira, no entanto, resume-se a envio de *pings*[7] ativamente entre os diversos processos existentes (TANENBAUM e VAN STEEN, 2007).

Entretanto, é desejável que falhas de desempenho possam ser tratadas, por isso, o módulo detector de falhas deve tolerar a hipótese de lentidão inesperada de um membro (TANENBAUM e VAN STEEN, 2007).

Por isso, a detecção de erros, quando baseada unicamente em um mecanismo de temporização, torna-se hábil em gerar falso-positivos, por isso, deve-se tentar remediar este equívoco proporcionando outras oportunidades de resposta às possíveis omissões observadas (BIRMAN, 2005). Uma possível solução para essa demanda seria a adoção de *gossiping* no qual cada nó anuncia periodicamente a seus vizinhos que ainda se encontra ativo e, em acréscimo, realiza a sondagem sobre os demais nós (VOGELS, 2003).

Outra característica importante para o desenvolvimento de qualquer subsistema de detecção de falhas seria a capacidade de distinguir inequivocamente uma falha de rede e de processo. Um modo de se alcançar isto seria descentralizando a decisão a respeito de qualquer falha observada no sistema. Visto que um nó aparentemente falho pode, na verdade, encontrar-se num enlace eventualmente lento ou defeituoso e, por isso, antes de se determinar qual o tipo de falha (rede ou processo) torna-se prudente consultar um processo vizinho e, dessa maneira, averiguar se, por outro enlace, o nó antes suspeito encontra de fato parado (TANENBAUM e VAN STEEN, 2007).

Salienta-se que no contexto do presente estudo a atividade de detecção de erros precisa contornar os desafios inerentes a um ambiente computacional distribuído e assíncrono, tendo em vista a possibilidade de coordenação e operação de dispositivos móveis e inteligentes no âmbito do espaço AVA.

Neste contexto, Fisher, Lynch e Paterson (1985) afirmam que num ambiente completamente assíncrono não há como se resolver de forma perfeita o problema do consenso, tal como a detecção de falha, mesmo sob a queda inesperada de apenas um único processo.

A propósito, cabe ressaltar que o problema do consenso em ambiente distribuído permite que os nós alcancem uma decisão comum a partir das mensagens recebidas mesmo sob a ocorrência de falhas. Algoritmos de consenso são utilizados para resolver muitos problemas práticos, tais como eleger um líder ou concordar sobre o valor de um sensor

[7] Ping: tipo de mensagem enviada para verificar a presença de uma entidade remota

replicado (CHANDRA e TOUEG, 1996).

O maior desafio neste cenário é que no sistema assíncrono não há como se delimitar, a priori, o tempo de transmissão dos dados nem tampouco a velocidade de processamento de qualquer dos nós. Assim, dizer que um sistema é assíncrono é, portanto, deixar de assumir qualquer pressuposto de ordem temporal. Por isso, neste ambiente não há como estabelecer limites para se determinar a exatidão acerca do estado do sistema, pois, na prática, a falta de sincronização entre os processos deriva de variações ambientais admitidas e de cargas inesperadas no sistema (CHANDRA, HADZILACOS e TOUEG, 1996).

Para contornar este desafio Chandra e Toueg (1996) resolvem propor uma abstração de detector de erro não-confiável em ambiente distribuído e assíncrono. Para os autores supramencionados, detector de erros não-confiável é um oráculo distribuído que provê sugestões, possivelmente incorretas, sobre qual nó do sistema pode, no momento, encontrar-se parado ou faltoso (*crash*). Cada processo correto (não faltoso) teria acesso local a um detector de erros não-confiável que seria responsável por monitorar o comportamento de outros processos e, dessa forma, constituir a relação de prováveis processos faltosos (lista de suspeitos) que sofre atualizações com o decorrer do tempo. Ou seja, decorrido certo tempo, se algum processo identificar que um nó, a princípio, suspeito deixou de ser suspeito (*cometimento de engano*) passará então a corrigir seu julgamento e passará a considerá-lo correto e, por isso, o retirará da lista de suspeitos.

Por fim, da consulta periódica ao módulo de detector de erros não-confiável ter-se-á a resolução do problema do consenso em ambiente distribuído e assíncrono (CHANDRA, HADZILACOS e TOUEG, 1996), tal como o pretendido no presente estudo, detecção de erros em sistema para AVA.

2.3.3.2. Detector de erros não-confiável em sistemas assíncronos

Conforme mencionado, detector de erros não confiável é um oráculo que provê informações sobre o estado dos processos em um sistema distribuído e assíncrono. A partir do subsistema local poder-se-á cometer enganos ao adicionar equivocadamente um processo correto na relação de processos suspeitos de terem falhado (CHANDRA e TOUEG, 1996).

Para melhor compreensão do que seja um detector de erros não confiável, consoante proposto por Chandra e Toueg (1996), faz-se necessário vislumbrar um sistema assíncrono como sendo uma coleção de n processos distribuídos p, $\Pi = \{p_1, p_2, ..., p_n\}$, onde $n > 1$.

Cada par de processos (p_1, p_2) é conectado por um canal de comunicação confiável. Para efeitos de simplificação assume-se a existência de um relógio global discreto, um dispositivo meramente abstrato para denotar a passagem de tempo que os processos p não possuem acesso. Ademais, adota-se a faixa de números naturais *T* para o contador de tempo.

O modelo de falha considerado por Chandra e Toueg (1996) é o de falha por parada, ou ainda, parada inesperada (*crash*). O modelo de falha *F* é uma função de *T* para 2^{Π}, onde *F(t)* denota o conjunto de processos que falharam por parada até o tempo *t*. Uma vez que o processo falha ele não se recupera, isto é, $\forall t: F(t) \subseteq F(t+1)$. O conjunto $\text{crashed}(F) = \cup_{t \in T} F(t)$ e $\text{correct}(F) = \Pi - \text{crashed}(F)$. Se $p \in \text{crashed}(F)$ diz-se que *p* parou em *F*, da mesma forma se $p \in \text{correct}(F)$ diz-se que *p* é correto em *F*. Por fim, assumem-se padrões de falha em que pelo menos um processo encontra-se correto, isto é, $\text{correct}(F) \neq \phi$.

Dado que o módulo detector de falhas *D* retorna um conjunto de processos que atualmente encontram-se suspeitos de parada, um histórico de detector de falhas *H* é uma função de $\Pi \times T$ para 2^{Π}. Dado *H(p,t)* o retorno do detector de falha de *p* no tempo *t*, diz-se que se $q \in H(p,t)$, logo *p* suspeita de *q* no tempo *t* em *H*[8]. Percebe-se, dessa forma, que dois processos distintos não necessitam concordar em sua respectiva lista de suspeitos, ou seja, se $p \neq q$ então se torna possível que $H(p,t) \neq H(q,t)$ (CHANDRA e TOUEG, 1996).

Informalmente, um detector de falhas *D* fornece um possível e talvez incorreto resultado do padrão de falhas *F* em um dado instante de execução *t* do sistema distribuído e assíncrono. Com maior rigor, *D(F)* é o conjunto de todos os históricos de detector de falhas que poderá ocorrer em um padrão de falhas *F* em cada detector de falhas *D*. Para não restringir *D* a uma implementação específica os autores Chandra e Toueg (1996) assumem duas propriedades abstratas que devem ser satisfeitas por qualquer *D*: completude e acurácia. A completude garante que processos que falham serão em algum momento declarados suspeitos enquanto acurácia delimita os enganos que possam ser cometidos (SENS, GREVE, *et al.*, 2008). Segundo Chandra e Toueg (1996), com apenas estes dois requisitos pode-se projetar e desenvolver aplicação tolerante a falha em sistema distribuído e assíncrono.

Segundo os mesmos autores, estas propriedades podem sofrer modulações, sendo duas no quesito completude e quatro na acurácia. Ou seja, a completude pode ser forte ou

[8] (CHANDRA e TOUEG, 1996) assumem a notação *H* como implícita quando óbvio o seu contexto.

fraca. Será forte quando cada processo que falha é permanentemente suspeito por todos os processos corretos (definição 1). Formalmente, *D* satisfaz a propriedade completude forte se: $\forall F, \forall H \in D(F),\ \exists t \in T,\ \forall p \in \text{crashed}(F),\ \forall q \in \text{correct}(F),\ \forall t' \geq t : p \in H(q, t')$. Por outro lado, *D* satisfará completude fraca quando eventualmente cada processo que falha torna-se permanentemente suspeito por algum processo correto (definição 2). Em termos formais, *D* satisfaz o quesito completude fraca se: $\forall F, \forall H \in D(F),\ \exists t \in T,\ \forall p \in \text{crashed}(F), \exists q \in \text{correct}(F),\ \forall t' \geq t : p \in H(q, t')$.

Conforme mencionado, a propriedade completude não é suficiente, haja vista que eventuais enganos precisam ser corrigidos, daí a necessidade de se prover acurácia em quatro níveis: forte, fraca, eventualmente forte e eventualmente fraca. A acurácia será forte quando nenhum processo será considerado suspeito antes que de fato tenha falhado (definição 3). Formalmente tem-se acurácia forte quando: $\forall F, \forall H \in D(F),\ \forall t \in T,\ \forall p, q \in \Pi - F(t) : p \notin H(q, t)$. É possível vislumbrar quão difícil será alcançar acurácia forte em muitos sistemas distribuídos e assíncronos reais, por isso tem-se a acurácia fraca que possibilita a situação de que pelo menos algum processo correto jamais se tornará suspeito (definição 4). Formalmente tem-se: $\forall F, \forall H \in D(F),\ \exists p \in \text{correct}(F),\ \forall t \in T,\ \forall q \in \Pi - F(t) : p \notin H(q, t)$. Sabendo-se que, na prática, este tipo de acurácia também se torna difícil de ser obtida, Chandra e Toueg (1996) defendem a possibilidade de detectores de falhas não confiáveis poderem suspeitar de dado processo, mesmo que erroneamente, pelo menos uma vez ou outra. Ou seja, admitem a possibilidade de satisfação eventual (ou casual) da acurácia, seja esta fraca ou forte. Tendo-se, por conseguinte, a acurácia eventualmente forte e a acurácia eventualmente fraca.

Para Chandra e Toueg (1996) a acurácia eventualmente forte determina que haja um momento no qual a acurácia forte finalmente possa ser estabelecida. Por consequência, saber-se-á que depois de um dado momento em que todos os processos defeituosos finalmente falharem os processos corretos serão aqueles: $\forall F, \exists t \in T, \forall t' \geq t : \Pi - F(t') = \text{correct}(F)$. Ou seja, a acurácia eventualmente forte defende que depois de um dado momento nenhum processo correto deverá ser considerado suspeito por qualquer outro processo correto (definição 5), em termos formais tem-se: $\forall F, \forall H \in D(F),\ \exists t \in T, \forall t' \geq t, \forall p,\ q \in \text{correct}(F) : p \notin H(q, t')$.

Por outro lado, acurácia eventualmente fraca pode ser compreendida como a possibilidade que em um dado momento algum processo correto não deva ser considerado

suspeito por qualquer outro processo correto (definição 6). Em termos formais seria: $\forall F, \forall H \in D(F),\ \exists t \in T, \exists p\ \in correct(F), \forall t' \geq t, \forall q \in\ correct(F) : p \notin H(q, t')$.

Com as propriedades estabelecidas podem-se vislumbrar as diversas classes de detectores de erros. Em princípio, um detector de erro da classe perfeita P é aquele que satisfaz simultaneamente as propriedades: completude forte e acurácia forte. Consoante discutido, em sistemas assíncronos reais este detector de falhas praticamente inexiste. Por isso, Chandra e Toueg (1996) estabelecem oito classes de detectores de erros derivadas da combinação pareada dos dois tipos de completude e dos quatro de acurácia, como se observa na Tabela 2.

Tabela 2 : Classes de detector de erro não-confiável (CHANDRA e TOUEG, 1996).

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\diamond\mathcal{P}$	*Eventually Strong* $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	*Weak* $\mathcal{W}$	$\diamond\mathcal{Q}$	*Eventually Weak* $\diamond\mathcal{W}$

Por meio de inferências construídas sobre proposições lógicas estabelecidas a partir de alguns algoritmos propostos, Chandra e Toueg (1996) demonstram formalmente que o problema do consenso distribuído pode ser resolvido em ambiente assíncrono usando qualquer detector de erros pertencente à classe eventualmente fraca $\diamond\mathcal{W}$ desde que o sistema detenha uma maioria de processos p corretos, $f < n/2$.

Contudo, os algoritmos de Chandra e Toueg (1996) não estão aptos às peculiaridades inerentes ao ambiente AVA. Destarte, buscaram-se trabalhos derivados da proposta anterior, tais como: Tai, Tso e Sanders (2004), Friedman e Tcharny (2005), Sridhar (2006) e destes identificou-se primeiramente em Greve, Sens, *et al.* (2011) e, em seguida, a atualização deste trabalho (GREVE, SENS, *et al.*, 2012), o paradigma ideal para desenvolvimento do módulo detector de erros objeto do presente estudo. Visto que estas últimas propostas endereçam satisfatoriamente as especificidades encontradas em um AVA, tais como mobilidade de processos e existência de membros desconhecidos.

2.3.3.3. Detector de erros em sistemas para Ambiente de Vida Assistida

A classe de detector de erros para redes sem fio com membros desconhecidos, $\diamond S^M$ (GREVE, SENS, *et al.*, 2011) é para nosso estudo a proposta mais adequada, visto que adapta às propriedades da classe eventualmente forte, $\diamond S$, às condições dinâmicas presente

em redes de sensores sem fio e em redes móveis auto organizáveis (MANETs[9]). Neste tipo de ambiente, tal como em AVA, a composição dos membros é desconhecida, bem como qualquer presunção global a respeito de número máximo de processos faltosos, presunção de completa conectividade bem como o estabelecimento de comunicação confiável entre os nós são completamente inverossímeis.

Recorde-se que, por definição, a classe de detectores de erros eventualmente forte, $\diamond S$, determina que todo processo que falha será em algum momento suspeito por todos os processos corretos (definição 1), além disto, haverá também um tempo depois do qual algum processo correto não deva ser considerado suspeito por qualquer outro processo correto (definição 6) (CHANDRA e TOUEG, 1996).

Dessa forma, Greve, Sens, *et al.* (2011) sugerem que a classe $\diamond S^M$ possui as condições mínimas para a resolução do problema de consenso de forma determinística em ambiente assíncrono e com mobilidade sem a necessidade de uso de temporizadores, pressupondo, no entanto, que haja uma maioria de processos corretos no sistema.

Destaca-se que a classe $\diamond S^M$ não se fundamenta no uso de temporizadores nem tampouco possui conhecimento a respeito da cardinalidade e da composição interna do sistema. Em contrapartida, o processo de detecção de erros ocorre apenas através da percepção local do nó acerca do estado do sistema e não da troca de variáveis globais.

O princípio básico de funcionamento do algoritmo proposto para a classe $\diamond S^M$ é a inundação entre nós vizinhos de suspeitas de falhas na rede (*gossiping*). De início, o nó detém apenas a percepção local acerca do estado do sistema e da rede. Em dado momento, à medida que um nó recebe e devolve informações aos nós vizinhos, por meio da troca de mensagens do tipo QUERY-RESPONSE, tem-se o estabelecimento de um consenso com demais nós corretos do sistema sobre as suspeitas de falhas no sistema.

Formalmente o sistema distribuído pode ser representado por um grafo G. A topologia de G é dinâmica, devido a ocorrência de junções arbitrárias, desconexões, paradas e movimentações. Em sistemas dinâmicos de redes desconhecidas, ao contrário de redes clássicas, os processos não conhecem Π, conjunto de todos os processos, nem a cardinalidade n deste conjunto. Ou seja, cada processo p_i conhece apenas um subconjunto V de Π, a respectiva vizinhança.

[9] MANETs (*Mobile ad hoc networks*) é um dos dois principais tipos de redes móveis. O outro tipo predominante são as redes móveis infraestruturadas (*Infraestructured networks*), composta de um grande número de nós móveis (*mobile hosts*, MH) e, relativamente poucas, mas robustas estações de suporte móvel (*mobile support stations,* MSSs). Neste outro tipo de rede os CHs comunicam-se somente com os MSSs e por estes são gerenciados (WU, CAO, *et al.*, 2007).

Portanto, $G = (V, E)$ e $(V \subseteq \Pi)$ onde E é o conjunto de ligações lógicas (canais de comunicação) entre os nós. Além disso, nenhuma hipótese temporal pode ser adotada com respeito às ações efetuadas pelos nós ou ainda pelos canais de comunicação, pois o sistema é de natureza assíncrona.

Ademais, neste estudo tem-se que o modelo de falha a ser considerado será a mera parada inesperada de processo (*crash*), tal que f é o número máximo de processos em que se tolera a movimentação ou mesmo a falha ($f < n/2$).

O valor de f deve ser de conhecimento de todos os nós do sistema. Para contornar o dilema de se estimar f, a implementação proposta por Greve, Sens, *et al.* (2012) considera que f deva ser inferior a cardinalidade dos nós vizinhos, v, ao invés de n. Ou seja, no contexto da vizinhança de p_i há f_i que deverá ser inferior a maioria dos processos corretos vizinhos, ou seja, $f_i < v_i / 2$. Sob esta visão, processo vizinho (ou conhecido) será todo aquele processo p_j que em dado momento enviou mensagem QUERY a p_i. Consequentemente, todo processo p_j integrará o conjunto $known_i$ (conhecidos) de p_i.

Quanto a forma de comunicação entre os processos o algoritmo de Greve, Sens, *et al.* (2011) adota o modelo broadcast *fair-lossy*, ou seja, mensagens podem ser perdidas, todavia, se p_i dissemina continuamente uma mensagem m para todos os processos vizinhos p_j, então, em dado momento, ou p_j receberá m ou então terá falhado. Além disso, os nós em Π podem se movimentar ou permanecer momentaneamente estáticos.

Quando um processo vizinho p_j se movimenta (passa a ser p_m) ele deixa de fazer parte de G, porém não notifica isso aos demais nós de V, pois também não sabe de antemão que mudou de localização (*passive mobility*). Dessa forma, do ponto de vista da vizinhança não é possível determinar se p_m movimentou-se, desconectou-se ou de repente falhou.

Quanto a perspectiva local de p_m, durante a alteração de vizinhança, o estado de p_m é preservado e posteriormente disseminado na nova (ou mesma) vizinhança V' em que se (re)estabelecer. Ademais, assume-se sempre que se um nó p_m é correto, logo, em um algum momento, irá se (re)conectar à rede (SENS, GREVE, *et al.*, 2008).

A proposta de implementação de detector de erros da classe $\diamond S^M$ segundo Greve, Sens, *et al.* (2011) encontra-se no Algoritmo 1 que descreve em passos o protocolo anterior. Em resumo, o processamento proposto é dividido em duas tarefas concorrentes: a) *Task* 1, responsável pela geração de suspeitas sob um laço infinito e b) *Task* 2, revoga suspeitas a partir de mensagens QUERY e propaga informações na rede por meio de RESPONSE.

Em trabalho posterior, Greve, Sens, *et al.* (2012) não modificaram a essência do Algoritmo 1, por isso, mantivemos este algoritmo, tendo em vista que nele há maior legibilidade da proposta apresentada.

Algoritmo 1 : Algoritmo original para implementação de detector de erros da classe $\diamond S^M$ (GREVE, SENS, *et al.*, 2011).

```
1  init:
2   susp_i ← ∅; mist_i ← ∅  ;  known_i ← ∅
3  Task T1:
4  Repeat forever
5     broadcast QUERY(susp_i, mist_i)
6     wait until RESPONSE received from ≥ α_i processes
7     rec_from_i ← all p_j, a RESPONSE is received in line 6
8     For all p_j ∈ known_i \ rec_from_i | ⟨p_j, −⟩ ∉ susp_i do
9        If  ⟨p_j, ct⟩ ∈ mist_i
10          Add(susp_i, ⟨p_j, ct + 1⟩)
11          mist_i = mist_i \ {⟨p_j, −⟩}
12       Else
13          Add(susp_i, ⟨p_j, 0⟩)
14 End repeat
15
16 Task T2:
17 Upon reception of QUERY (susp_j, mist_j) from p_j do
18 known_i ← known_i ∪ {p_j}
19 For  all ⟨p_x, ct_x⟩ ∈ susp_j do
20 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
21    If p_x = p_i
22       Add(mist_i, ⟨p_i, ct_x + 1⟩)
23    Else
24       Add(susp_i, ⟨p_x, ct_x⟩)
25       mist_i = mist_i \ {⟨p_x, −⟩}
26 For  all ⟨p_x, ct_x⟩ ∈ mist_j do
27 If ⟨p_x, −⟩ ∉ susp_i ∪ mist_i or (⟨p_x, ct⟩ ∈ susp_i ∪ mist_i and ct < ct_x)
28    Add(mist_i, ⟨p_x, ct_x⟩)
29    susp_i = susp_i \ {⟨p_x, −⟩}
30    If (p_x ≠ p_j)
31       known_i ← known_i \ {p_x}
32 send RESPONSE to p_j
```

Analisando o Algoritmo 1 observa-se que a mensagem QUERY é enviada a todos os nós da rede por meio da técnica de difusão (*broadcast*). Na mensagem disseminada tem-se o estado local: $susp_i$ (suspeitos) e $mist_i$ (enganos) do detector de erros D de p_i. Em seguida, o processo p_i aguarda por α_i RESPONSES dos vizinhos (incluindo o próprio) para atualizar o estado local e D. O alcance do limiar α_i encerra o ciclo QUERY-RESPONSE e, por conseguinte, descarta a utilização de qualquer temporizador. Para rotular as informações detectadas de suspeitas e enganos na rede é utilizado um contador (*ct*), inicialmente em zero, e incrementado para cada nova suspeita levantada. Este contador (*ct*) serve apenas de parâmetro para definir o grau de atualização da informação registrada, ou seja, quanto maior for, mais atual é a informação relacionada. Por fim, a lista auxiliar $known_i$ registra o conhecimento dos processos vizinhos e rec_from_i os nós que, por ventura, devolveram

RESPONSE a última mensagem QUERY de p_i.

Ademais, para se cumprir os objetivos almejados pelo Algoritmo 1 (GREVE, SENS, *et al.*, 2011) há a necessidade de o sistema apresentar um mínimo de condições de estabilidade, caracterizada, por exemplo, pela ausência de conectividade curta, reingresso constante e movimentações rápidas de processos, que deverão ser satisfeitas pelo período de tempo suficiente para a completa execução da computação proposta. Neste sentido, tem-se que:

a) processo vizinho p_j (*membership*) é todo aquele que se torna conhecido (envia QUERY a p_i) enquanto p_i encontra-se correto e estável;
b) conectividade estável, representada pelo grafo $G(S) \subseteq G$, é a presunção de que haverá pelo menos uma ligação (meio de comunicação) estável entre cada par de processos vizinhos $(p_i, p_j) \in G(S)$;
c) um ciclo estável QUERY-RESPONSE (QR) deve possuir as seguintes propriedades:
    i. *QR-Vality*: Se um QUERY é recebido por p_j, então foi enviado por um p_i;
    ii. *QR-Uniformity*: Um QUERY é recebido pelo menos uma vez por um processo p_j do sistema;
    iii. *QR-Stable-Termination*: Se um processo p_i é correto (não falhou nem deixou a vizinhança) enquanto submetia QUERY por difusão aos vizinhos, logo deverá receber pelo menos α_i RESPONSES (incluindo o do próprio p_i).
d) o valor α_i será uma variável local que estabelece a quantidade mínima de processos corretos, calculada a partir da seguinte fórmula: $v_i - f_i$. Onde f_i representa a quantidade máxima de processos vizinhos faltosos[10]: $f_i = v_i / 2 - 1$;

Greve, Sens, *et al.* (2012) afirmam que com a satisfação dos requisitos anteriores o Algoritmo 1 proporcionará a satisfação das seguintes propriedades:

a) *Stable Termination Property* (***SatP***): determina que se uma QUERY é submetida à rede por p_i logo será recebida por pelo menos um processo

[10] Este critério segue os recentes trabalhos de tolerância a falhas em rede de comunicação sem fio. Neste modelo a tolerância a falhas prioriza o aspecto local ao invés do global, tendo em vista o dinamismo e a falta de confiabilidade dos canais de comunicação via rádio (GREVE, SENS, *et al.*, 2012).

estável e vizinho p_j diferente de p_i. Esta propriedade garante a completude e a acurácia pretendida pela classe $\diamond S^M$;

b) *Stabilized Responsiveness Property* (***SRP***): diz que eventualmente em dado instante *t''* qualquer processo p_i, que tenha recebido um RESPONSE de um outro processo vizinho e estável p_j ($p_i \neq p_j$) em instante anterior *t'* (*t'* < *t''*), deve, portanto, possuir uma mensagem RESPONSE de p_j em seu conjunto rec_from_i, ou então, p_i já se encontrava parado em momento anterior *t* (*t* < *t'* < *t''*). O cumprimento desta propriedade preserva a acurácia, visto que, em razão da falta de um critério temporal, p_j poderia indevidamente se tornar um processo suspeito de parada (*crash*).

Adicionalmente, Greve, Sens, *et al.* (2011) garantem que se terá também a realização da seguinte propriedade:

c) *Mobility Property* (***MobiP***): disciplina que todo nó móvel p_m em um dado instante *t* alcança uma (nova) vizinhança, e, por isso, deve receber mensagem QUERY de pelo menos um nó estável p_j ($p_j \neq p_m$) em algum momento posterior *t'*, além do próprio QUERY. Essa propriedade é resultante do requisito dos canais de comunicação serem *fair-lossy*, ou então, tem-se que p_m encontrar-se-á parado;

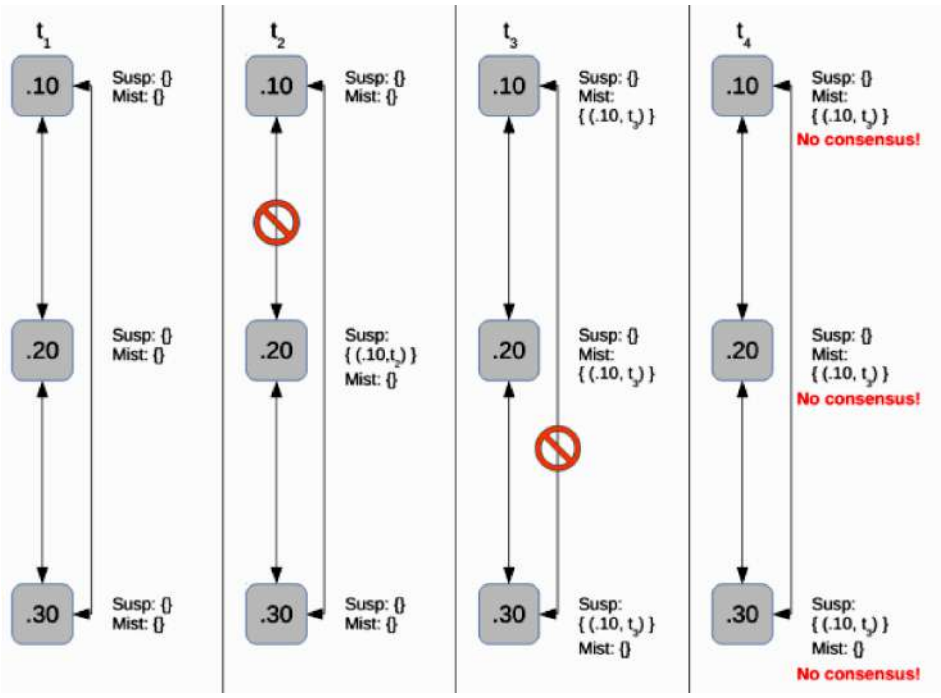
Assim, o desenvolvimento de qualquer detector de erros da classe $\diamond S^M$ deve considerar a manutenção dos pressupostos elencados de modo a se prover a satisfação das propriedades supramencionadas. Seja na definição da topologia da rede, por exemplo, através da existência de nós estacionários (estações base ou *gateway*), quanto na especificação de parâmetros tal como α_i que devem ser obtidos em tempo de execução a depender do comportamento da rede.

Destarte, percebendo a aplicação do Algoritmo 1 no contexto de sistemas para AVA, desenvolveu-se um módulo de detecção de erros sob a linguagem Java™ para incrementar a confiabilidade da plataforma AVA escolhida, OpenHAB mais Mosquitto. Em síntese, as tarefas T1 e T2 do Algoritmo 1 foram transformadas em threads Java™, além disso, utilizou-se o protocolo de transporte UDP com uma pequena adaptação para se confirmar o recebimento de mensagem QUERY. A confirmação de entrega sob o protocolo UDP visou tão somente identificar qual(is) nó(s) vizinho(s) poder-se-ia(m) esperar a devolução de mensagem(ns) RESPONSE.

Após o desenvolvimento do módulo detector de erros ingressou-se na fase de testes de execução do sistema em um protótipo construído para simular a aplicação do sistema em espaço AVA. Neste período, observou-se que durante o início da execução do módulo de detecção de erros ocasionalmente ocorria um problema de falta indevida de consenso entre os nós do sistema acerca da definição de suspeitas de parada (*crash*) na rede, comprometendo, portanto, a exatidão da propriedade acurácia eventualmente fraca especificada para a classe $\diamond S^M$.

Objetivamente, verificou-se que, na hipótese de no mínimo dois e sucessivos rompimentos de enlace de rede, havia a geração simultânea no sistema, e, portanto, sob o mesmo rótulo (*ct*), de informações de suspeita e engano acerca de um mesmo nó p_j. Em virtude disso, o Algoritmo 1 não podia revogar uma falsa suspeita de parada de um nó correto p_j comprometendo, portanto, o grau de acurácia quando do restabelecimento dos enlaces anteriormente defeituosos. Para ilustrar este cenário tem-se na Figura 9 uma representação no tempo (raias t_1, ... t_4) deste fato para um conjunto de três nós vizinhos, representados por identificadores: .10, .20 e .30.

Figura 9 : Suposto cenário de falta indevida de consenso do detector de erros da classe $\diamond S^M$.



Depois de uma depuração do módulo identificou-se que a causa do problema poderia ser encontrado no operador relacional existente na parte final dos testes presentes nas linhas 20 e 27 do Algoritmo 1 ([...] and $ct < ct_x$). Para solucionar o defeito bastaria trocar o

comparador menor (<) por menor ou igual (<=), ficando, portanto, assim: [...] and $ct <= ct_x$. Consequentemente, após esta alteração restabelecer-se-ía o consenso devido promovendo a satisfação da acurácia eventualmente fraca especificada para a classe $\diamond S^M$. Cabe salientar que em trabalho posterior, (GREVE, SENS, *et al.*, 2012), mantém-se o operador (<) conforme descrito no Algoritmo 1.

Poder-se-ia cogitar que o problema apontado seria em virtude do descumprimento do requisito conectividade estável. Todavia, como se trata do desenvolvimento de um caso de uso, módulo detector de erros de sistema para Ambiente de Vida Assistida, as falhas de rede inevitavelmente aparecem, especialmente os particionamentos de rede ou deslocamento de nós para/entre subredes distintas. E é exatamente por isso devem ser analisadas, tratadas e corrigidas. Pois, consoante Jalote (1994) sistemas distribuídos frequentemente possuem em sua estrutura a presença de nós e rede de comunicação como componentes básicos do sistema, por isso, qualquer ocorrência de falha nestes dois elementos precisa ser avaliada quando do monitoramento do estado do sistema.

Entretanto, após uma reanálise cuidadosa do pressuposto problema, especialmente, após a leitura do trabalho (GREVE, SENS, *et al.*, 2012) verificou-se que esta falta de consenso deve-se a uma falha de interpretação do Algoritmo 1, especificamente na linha 10. A implementação inicial atribuía à variável *ct* um valor incremental local e arbitrário (p. ex. contador de ciclos QUERY-RESPONSE) e não o incremento do valor *ct* anteriormente armazenado na lista local de enganos ($mist_i$). Por isso, feita a devida correção e submetendo o módulo de detecção de falhas a novos testes verificou-se, enfim, que este possível defeito concretamente inexiste, pois as suspeitas passaram a ter valores *ct* par e os enganos, ímpar.

Em continuidade dos testes pós-desenvolvimento detectou-se, enfim, a existência de um verdadeiro problema no comportamento do Algoritmo 1, pelo menos no contexto de sistema para AVA: a possibilidade de um nó p_i se autodeclarar suspeito sem conseguir, posteriormente, modificar esta avaliação quando da confirmação do engano cometido (avaliação de mensagem QUERY com p_i em $susp_i$ por p_i). A origem deste problema encontra-se na linha 8 do Algoritmo 1, pois o mesmo não exclui p_i da geração de suspeitas locais, logo, caso a própria mensagem RESPONSE não figure dentre as mensagens recebidas (rec_from_i) haverá a autodeclaração de suspeita (possível detecção de falha *loopback*). Esta situação pode até ser admitida por um dado momento, porém o que nos parece inaceitável é a falta de tratamento posterior no próprio Algoritmo 1 visando a

correção desta equivocada avaliação quando enfim houver o retorno de uma mensagem RESPONSE do próprio p_i possibilitando essa devida correção.

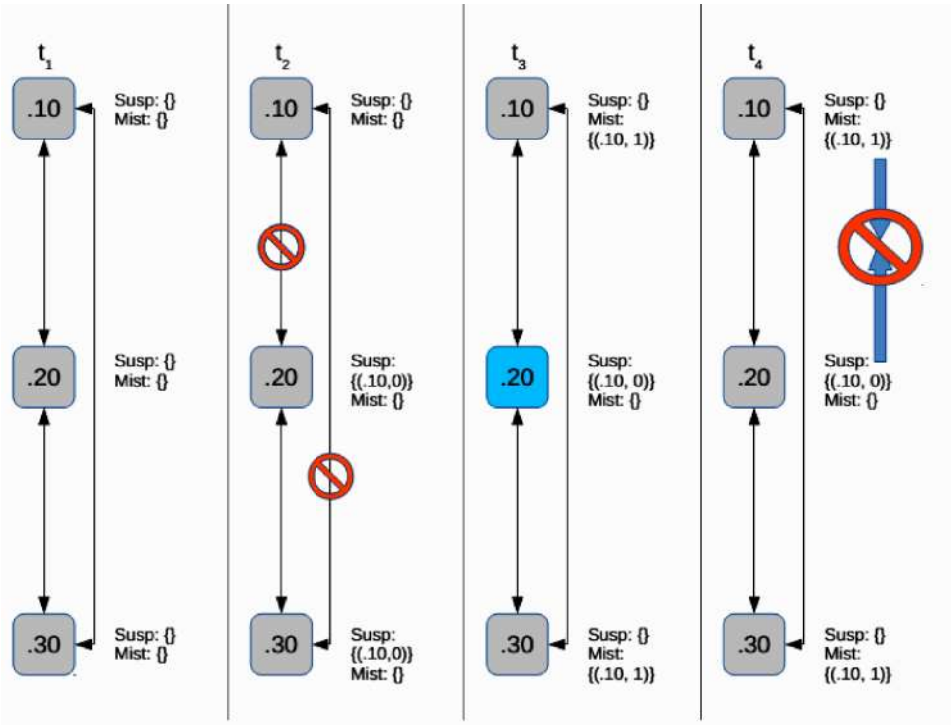
Salienta-se que não se encontrou qualquer comentário ou recomendação elidindo ou justificando a existência deste comportamento em trabalhos relacionados: (GREVE, SENS, *et al.*, 2011) e (GREVE, SENS, *et al.*, 2012). Tem-se, portanto, a existência de um problema no Algoritmo 1 que precisa ser reparado pelo menos no que se refere ao cenário de Ambiente de Vida Assistida. A modificação necessária, neste particular, podem ser de duas formas, a primeira é bastante simples, basta acrescentar uma nova condição na linha 8 do Algoritmo 1 excluindo o próprio nó p_i na geração de suspeitas locais ($susp_i$), ficando, portanto, a linha 8 assim: ... | ($p_j \neq p_i$) *do*. Esta primeira opção evita, portanto, a detecção de falhas *loopback* que em no contexto deste trabalho torna-se completamente indiferente, pois na configuração do sistema utilizar-se-á nós redundantes visando à implementação de mascaramento de falhas do tipo parada (*crash*). Ou seja, a detecção de indisponibilidade local é irrelevante para o conjunto do sistema, pois o que se espera é que um conjunto máximo *f* de falhas ocorra dentro de todos os sistemas.

Em acréscimo, a segunda possibilidade de alteração seria a irrestrita eliminação de suspeita acerca do próprio nó, ou seja, deve-se ter como desejável que o nó deve obrigatoriamente eliminar a suspeição sobre si em qualquer mensagem QUERY que por ventura vier a receber independentemente do valor constante na variável ct_x. Esta segunda forma de modificação também pode resolver outro problema posteriormente identificado, conforme será adiante detalhado.

Em avaliações posteriores verificou-se também a circunstância de atraso de um nó p_j na disseminação (ou repasse) de suspeita acerca de p_i comprometendo a capacidade do nó p_i corrigir tal equivocada suspeita. Este problema encontra-se localizado na linha 20 do Algoritmo 1, pois este exige sempre que qualquer informação de suspeita acerca de p_i disseminada na rede para ser devidamente corrigida (linhas 21 e 22) precisa obrigatoriamente ser mais recente ($ct < ct_x$) que uma suspeita ou um engano anteriormente registrado ($susp_i$ U $mist_i$) no próprio p_i. Para corrigir tal comportamento, faz-se necessário o deslocamento dos comandos existentes nas linhas 21 e 22 para antes do teste presente na linha 20 do Algoritmo 1.

Para ilustrar este último problema identificado tem-se na Figura 10 uma representação deste cenário para um conjunto de três nós vizinhos (.10, .20 e .30).

Figura 10 : Cenário de propagação de suspeita defasada (0) em colisão com engano atualizado (1) em nó .10.



Por fim, um desafio prático enfrentado pelo presente estudo quanto aos comandos previstos no Algoritmo 1 foi a manutenção de estado da variável $known_i$ em nós pertencentes a novas sub-redes (derivadas de particionamento) dotadas de mais recente cardinalidade v_i', por exemplo, inferior ao limiar anteriormente estabelecido α_i (v_i / 2 + 1). Para contornar tal situação, e seguindo a recomendação de sempre se obter α_i a partir da (nova) vizinhança e da dinamicidade própria de redes MANETs (GREVE, SENS, *et al.*, 2012), alterou-se o comportamento do Algoritmo 1 para que atingido um limiar arbitrário de ciclos QUERY-RESPONSE malsucedidos (quantidade de mensagens RESPONSE inferior a α_i) reinicie-se o estado de $known_i$ para vazio. Dessa forma, e a partir de um novo contexto (v_i') reavaliar-se-ía qual deva ser o valor adequado para α_i (v_i' / 2 + 1).

Diante de todas estas considerações supramencionadas fez-se necessário esboçar um Algoritmo 2 no qual se materializa as modificações resultantes das soluções dos problemas e desafios identificados na fase de testes pós-desenvolvimento do módulo de detecção de falhas em sistema para AVA originalmente inspirados no Algoritmo 1 proposto por Greve, Sens, *et al.* (2011) e, posteriormente, revalidado por Greve, Sens, *et al.* (2012).

Algoritmo 2 : Alterações no Algoritmo 1 para desenvolvimento de módulo detector de falhas em sistema para AVA

```
01  init:
02  susp_i ← ∅; mist_i ← ∅; known_i ← ∅; QR_sucess_i ← false; qt_QR_unsucess_i ← 0; LIMIT ← N
03  Task T1:
04  Repeat forever
05          broadcast QUERY (susp_i, mist_i)
06          Repeat until QR_sucess_i or qt_QR_unsucess_i > LIMIT
07                  If RESPONSE received from >= α_i processes
08                          QRsucess_i ← true
09                  Else
10                          qt_QR_unsucess_i ← qt_QR_unsucess_i + 1
11          If qt_QR_unsucess_i > LIMIT
12                  known_i ← ∅; qt_QR_unsucess_i ← 0
13          Else
14                  rec_from_i ← all p_j, a RESPONSE is received in line 7
15                  For all p_j in (known_i - rec_from_i) and <p_j, -> not in susp_i and (p_j ≠ p_i) do
16                          If <p_j, ct> in mist_i
17                                  Add(susp_i, <p_j, ct + 1>)
18                                  Remove(mist_i, <p_j, ->)
19                          Else
20                                  Add(susp_i, <p_j, 0>)
21  End Repeat
22
23  Task T2:
24  Upon reception of QUERY (susp_j, mist_j) from p_j do
25  Add(known_i, p_j)
26  For all <p_x, ct_x> in susp_j do
27          If (p_x = p_i)
28                  Add(mist_i, <pi, ct_x + 1>)
29          Else If <p_x, -> not in (susp_i U mist_i) or (<p_x, ct> in (susp_i U mist_i) and ct < ct_x)
30                  Add(susp_i, <p_x, ct_x>)
31                  Remove(mist_i, <p_x, ->)
32  For all <p_x, ct_x> in mist_j do
33          If <p_x, -> not in (susp_i U mist_i) or (<p_x, ct> in (susp_i U mist_i) and ct < ct_x)
34                  Add(mist_i, <p_x, ct_x>)
35                  Remove(susp_i, <p_x, ->)
36                  If (p_x ≠ p_j)
37                          Remove(known_i, p_x)
38  send RESPONSE to p_j
```

2.3.3.4. Módulo de tolerância a falhas em sistema para AVA

Para avaliar a eficácia do módulo detector de erros precisou-se também desenvolver um subsistema de tolerância a falhas de modo a comprovar-se a realização da completude e da acurácia pretendida pela classe $\diamond S^M$.

Cabe ressaltar, portanto, que não é escopo do presente trabalho propor e especificar a mais adequada forma de solução de tolerância a falhas em sistema para AVA. Por isso, projetou-se um módulo de tolerância a falhas que simplesmente elege um eventual nó líder de agrupamento ou vizinhança (*Cluster Header*, CH) para recepcionar todos os dados coletados pelos demais nós do agrupamento bem como gerenciá-los. Conforme Wu, Cao *et al.* (2007), a estratégia de agrupamento tem sido amplamente adotada em MANETs haja vista ser a que apresenta: melhor eficiência para troca de mensagens, maior estabilidade e ser a mais favorável para crescimento da rede em escala (*scalability*).

Algumas estratégias mais sofisticadas de definição de liderança poderiam ser adotadas, a exemplo de Wu, Cao *et al.* (2007) e Cao, Raynal *et al.* (2007). Todavia, por questões de simplificação utilizou-se neste trabalho um processo mais rudimentar. O critério adotado de eleição de novo líder de agrupamento resume-se ao encontro do vizinho

ativo de menor identificador de rede (endereço IP).

Em todos os nós do sistema (computadores de placa única) tem-se em funcionamento a plataforma AVA com idênticas configurações, com o fito de se aplicar a estratégia de redundância de software para mascaramento de falhas em plataforma AVA. Na hipótese de falha do nó líder, ou ainda, o mero cometimento de suspeita acerca dele, produzir-se-á o reinício do processo de eleição de nó líder em cada processo correto do sistema. Consequentemente, ter-se-á o redirecionamento das informações coletadas por todos os processos distribuídos, menos, por óbvio, do novo líder. Isto porque ao eventual nó líder competirá recepcionar todas as informações, incluindo as listas de suspeitas e enganos de todos os demais nós do agrupamento, bem como gerenciar o espaço AVA como um todo.

Cada nó do sistema possui um ou mais dispositivos embarcados acoplados. Cada dispositivo embarcado é composto por um microcontrolador gerenciando um sensor ou atuador. Os dispositivos embarcados quando iniciados ligam-se ao nó (mestre) do sistema que primeiro responder ao pedido de vinculação. Após o encontro deste nó mestre há o envio dos dados encontrados bem como se chancela a subordinação do dispositivo embarcado (escravo) para a execução de comandos no âmbito do espaço AVA.

Quanto a aplicação cliente, OpenHAB *Client*, tem-se também, no processo de carga (ou configuração), a mesma eleição de nó líder de agrupamento que ocorre com os processos do sistema. Após este passo inaugural, o cliente monitora as informações disponíveis na plataforma AVA do nó líder de agrupamento bem como a este submete as ações que precisam ser executadas.

Concluída a fase de desenvolvimento fez-se a execução experimental do módulo de detecção e tolerância a falhas com vistas a avaliar a efetividade destas medidas para o incremento da dependabilidade do sistema para Ambiente de Vida Assistida.

2.4. Trabalhos correlatos

Há raros estudos correlatos na área de sistemas distribuídos assíncronos. O estudo de Jiménez, Arévalo e Fernández (2006), por exemplo, para propor um detector de erros para sistemas de membros desconhecidos assume pressupostos temporais tais como a necessidade de canais de comunicação temporizados, situação totalmente imprópria para o contexto enfrentado pelo presente trabalho. Ao final, Jiménez, Arévalo e Fernández (2006) comprovaram as proposições por meio da aplicação da lógica formal.

Greve, Sens, *et al.* (2011) e Greve, Sens, *et al.* (2012) propuseram a classe de detector de erros para redes sem fio com membros desconhecidos, $\diamond S^M$. Esta classe possui os requisitos adequados para aplicação no contexto de Ambiente de Vida Assistida, a saber: a permissão para mobilidade dos nós bem como a possibilidade de haver nós desconhecidos. Todavia, o detector de erros proposto não foi esboçado estritamente para o cenário de AVA, por isso, conforme detalhado em subseção anterior (2.3.3.3), a implementação do Algoritmo 1 apresentou algumas restrições práticas que precisaram ser ajustadas para melhor adequar-se às necessidades deste estudo. Ademais, Greve, Sens, *et al.* (2012) limitaram-se a avaliar o desempenho do módulo detector de erros proposto em simulador OMNeT++. Ao final, Greve, Sens, *et al.* (2012) apresentaram algumas análises estatísticas acerca dos resultados obtidos em simulador.

Quanto a estudos específicos na área de AVA o que se aproxima do presente é o de Bruckner, Yin e Faltinger (2012). Este trabalho realizou um experimento controlado em quatro localidades diferentes num intervalo de quatro meses. A pesquisa além de tratar dos aspectos de comissionamento facilitado em espaço AVA também se preocupou com os requisitos de qualidade do sistema de controle, por exemplo, ressaltando a falta de exatidão nos dados emitidos por sensores. Para contornar este problema recomendam o uso da técnica de fusão de dados em sensores replicados.

Entretanto, Bruckner, Yin e Faltinger (2012) não propuseram nem desenvolveram um módulo de detecção de erros, apenas limitaram-se a aplicar a técnica de mascaramento de falhas. Por isso, pode-se dizer o presente estudo pode vir a preencher tal lacuna ao estudar e propor um módulo detector de falhas em Ambiente de Vida. A seguir, tem-se a Tabela 3 com a comparação[11] dos trabalhos correlatos frente ao presente estudo:

Tabela 3 : Comparação dos estudos correlatos frente a proposta do corrente estudo

Estudo	Voltado para o contexto de AVA	Realização de experimento	Análise estatística dos resultados	Módulo detector de falhas
Jiménez, Arévalo e Fernández (2006)	N	N	N	S
Greve, Sens, et al. (2011) e Greve, Sens, et al. (2012)	N	S[12]	S	S
Bruckner, Yin e Faltinger (2012)	S	S	S	N
Este estudo	S	S	S	S

[11] A letra **N** representa que o estudo referenciado **não contemplou** tal característica enquanto a letra **S** sinaliza que a mesma característica foi **contemplada satisfatoriamente**.

[12] O experimento efetuado neste caso resume-se a execução do módulo detector de erros proposto em simuladores computacionais.

3. Metodologia

Para compor um processo de pesquisa sobre de técnicas automáticas de detecção e tratamento de falhas em sistema para Ambiente de Vida Assistida com o emprego de tecnologias abertas, fez-se necessário recorrer a uma revisão sistemática dos autores que tratam, validam ou questionam a problemática.

Com base nos pressupostos teóricos, foi efetuada a pesquisa de plataformas AVA bem como o emprego de tecnologias abertas (*open hardware/software*) em sistema para AVA. Definida a plataforma a ser aprimorada e as tecnologias abertas a serem utilizadas, foi projetado, desenvolvido e validado um módulo responsável pelo levantamento das falhas em um contexto de sistema para Ambiente de Vida Assistida.

Em seguida, foi elaborado um protótipo de Ambiente de Vida Assistida que possibilitasse avaliar os diversos cenários que pudessem comprometer a dependabilidade do sistema para AVA. Por fim, pretendeu-se, com esse processo de pesquisa, estudar, analisar, discutir e identificar a melhor forma de desenvolver medidas de detecção de falhas em sistema para AVA com o emprego de tecnologias abertas.

3.1. Materiais e Métodos

Para realização do proposto na Metodologia, precisou-se elaborar um experimento controlado que validasse o funcionamento do módulo de detecção de erros especificado e desenvolvido na presente pesquisa.

Nesta e nas duas próximas seções, o trabalho é apresentado como um processo experimental. O mesmo segue as diretrizes de Wohlin, Runeson *et al.* (2000). Esta seção irá focar na definição do objetivo e no planejamento do experimento.

3.1.1. Definição do objetivo do experimento

O objetivo desta seção é avaliar, por meio de um experimento controlado, o percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede num protótipo de Ambiente de Vida Assistida sob duas implementações distintas de detectores de erros da classe $\diamond S^M$: a primeira será a versão original do Algoritmo 1, a segunda, será uma versão modificada, Algoritmo 2.

Para melhor compreensão, entende-se por falsa suspeita de parada a situação em que um

nó equivocadamente assume e mantém como falho outro nó de fato correto (definição 7). Em outros termos, seria o cometimento persistente e indevido de um óbvio engano (*Mistake*).

O objetivo do experimento foi formalizado através do modelo GQM proposto por Basili e Weiss (1984): O experimento visa analisar a eficácia obtida acerca da definição da lista de suspeitas de falha ($susp_i$) na rede, com a finalidade de avaliar a discrepância existente entre duas implementações diferentes de detector de erros da classe $\diamond S^M$ (a primeira versão original e a segunda, modificada), com respeito ao percentual de falsa suspeita de parada (percMistakes), do ponto de vista dos profissionais da área de dependabilidade de sistemas, no contexto de tolerância a falhas em sistema para Ambiente de Vida Assistida.

3.1.2. Planejamento do experimento

3.1.2.1. Formulação de hipóteses

A questão de pesquisa para o experimento que precisa ser respondida é a seguinte: qual implementação proporciona maior eficácia quanto à definição de lista de suspeitas de falha? Para avaliar esta questão foi utilizada a seguinte métrica: percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede. Em outros termos, seria o equivalente a aferir pela via complementar o grau de satisfação da acurácia eventualmente fraca obtido pelo detector de erros da classe $\diamond S^M$, sob duas implementações distintas: original (Algoritmo 1) e modificada (Algoritmo 2). Desse modo, tendo o objetivo e métrica definida, serão consideradas as seguintes hipóteses:

HIPÓTESE 1

- H_0: O detector de erros para Ambiente de Vida Assistida quando implementado em qualquer uma das versões: original (Algoritmo 1) e modificada (Algoritmo 2), oferece similar percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede ($\mu_{\%Original} = \mu_{\%Modificada}$);

- H_1: O detector de erros para Ambiente de Vida Assistida em sua implementação original (Algoritmo 1) oferece maior percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede quando comparada à versão modificada (Algorimo 2) ($\mu_{\%Original} > \mu_{\%Modificada}$).

Para a Hipótese 1 formulada, a H_0, é a hipótese que se deseja rejeitar. Para averiguar

quais das hipóteses serão rejeitadas, será considerada variável dependente: percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede sob implementações distintas (Algoritmo 1 e Algoritmo 2) do módulo detector de falhas em sistema para AVA. E serão consideradas variáveis independentes: os dispositivos inteligentes em rede (*Networked Artifacts* SW/HW) e a plataforma para Ambiente de Vida Assistida (*AAL Platform*), OpenHAB mais Mosquitto.

3.1.2.2. Variáveis independentes

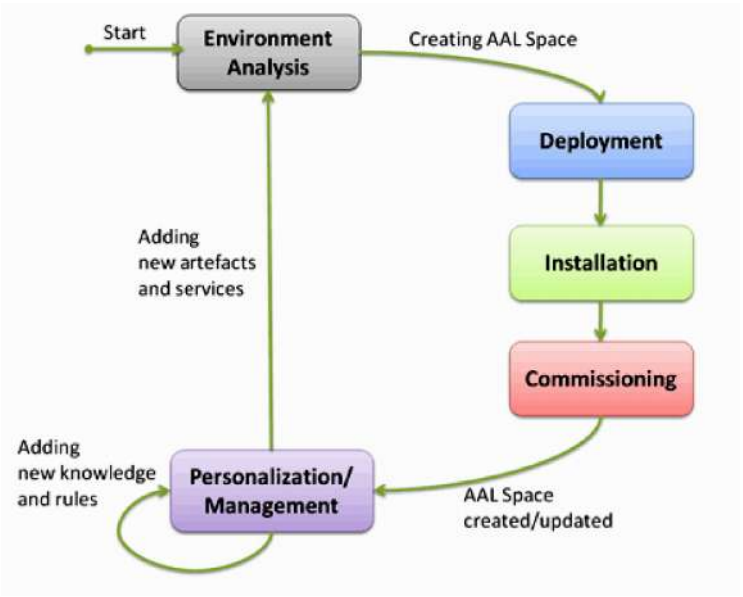
A seguir, será descrita apenas a seguinte variável independente do experimento: dispositivos inteligentes em rede (*Networked Artifacts* SW/HW), pois em subseção anterior (2.3.2) já se discutiu a segunda variável independente, ou seja, a plataforma AVA utilizada para o desenvolvimento do experimento.

Consoante mencionado anteriormente, sistemas para AVA proveem serviços às pessoas que necessitam de cuidados especiais e, para tanto, são utilizados dispositivos inteligentes em rede (sensores e atuadores dirigidos por microcontrolador) para materializar a execução destas facilidades. A orquestração dos dispositivos inteligentes em rede é efetuada pela plataforma AVA que materializa uma arquitetura de referência própria com vistas a possibilitar o compartilhamento de recursos, a conscientização do contexto e a personalização da assistência a ser prestada a pessoa idosa ou com mobilidade reduzida (FURFARI, TAZARI e EISEMBERG, 2011).

Espaços AVA são ambientes inteligentes centrados em seres humanos. Os dispositivos incorporados em tais ambientes operam coletivamente compartilhando informação e inteligência que são providas pela plataforma AVA que os interconecta. A construção de um ambiente AVA inicia pela análise do meio, segue com a definição dos recursos e dispositivos inteligentes a serem empregados e termina com a instalação e o comissionamento destes artefatos.

Quando o sistema se encontra em fase de operação, tem-se o gerenciamento (aplicação de regras e geração de conhecimento), ou, eventualmente, a personalização com a adição de novos serviços e artefatos. A Figura 11 descreve esquematicamente todo este processo (FURFARI, TAZARI e EISEMBERG, 2011).

Figura 11 : Processo de construção de um espaço AVA (FURFARI, TAZARI, & EISEMBERG, 2011).



Orientado por tal processo e pesquisando-se em fóruns de usuários e desenvolvedores de ambientes inteligentes, neste estudo optou pela utilização de dois tipos de dispositivos de *hardware*:

a) computador de placa única (*single-board computer*), dentre os quais se destaca o Raspberry Pi e seus modelos Zero, 2 e 3, para atuar como um elemento *gateway* do sistema para AVA;

b) dispositivo embarcado para Internet das coisas (*Internet of Things*, IOT) NodeMCU ESP8266 para realizar a comunicação, o sensoriamento e a execução de comandos sobre o espaço AVA (BRUCE, 2015).

A abordagem adotada encontra respaldo na proposta delineada por Bruckner, Yin e Faltinger (2012) quando defendem a estrutura combinada de nós acoplados a um ou mais sensores para comissionamento facilitado de Ambiente de Vida Assistida.

O computador de placa única (*single-board computer*) Raspberry Pi bem como seus múltiplos variantes (Banana Pi, Orange Pi etc) são verdadeiros PCs do tamanho de um cartão de crédito. Trata-se de pequenos e poderosos computadores capazes de serem empregados em diversos projetos de automação, além de servir para muitas outras tarefas que um computador pessoal tradicional pode desempenhar, tais como: elaboração de planilhas, processamento de texto, acesso à Internet e entretenimento. Este equipamento,

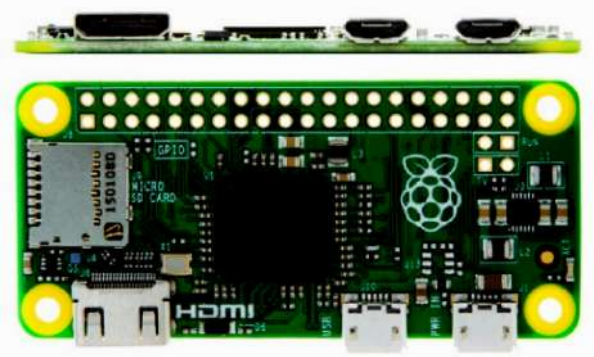
embora tenha tamanho reduzido, possui recursos de processamento capaz de reproduzir som e vídeo em alta definição.

O objetivo da Raspberry Pi Foudation quando criou este minúsculo computador era disseminar mundialmente a computação para adultos e crianças, seja através da programação ou da montagem de projetos eletrônicos. Tal propósito somente seria possível quando a aquisição de um computador não custasse algo além de algumas dezenas de dólares (RASPBERRYPI-FOUNDATION, 2017).

Em nosso estudo foram utilizadas algumas versões do Raspberry Pi (Zero, 2 e 3), como também dos clones: Banana Pi (M1) e Orange Pi (Zero e One). De todos estes dispositivos, verificou-se que o Raspberry Pi Zero foi o que apresentou a melhor relação de custo/benefício para o cumprimento do propósito deste estudo, haja vista ser a versão mais barata (custo a partir de U$5 até U$35) além de ser o que consume menos energia (no máximo 0,35 Amps) (RASPBERRYPI-FOUNDATION, 2017). Todavia, em todos foi possível executar uma versão do sistema operacional Linux com acesso à rede Ethernet bem como o framework Java/OSGi acompanhado da plataforma AVA definida para este estudo.

A especificação técnica do Raspberry Pi Zero lançado em novembro de 2015 é composta de: 1 processador de aplicação Broadcom BCM2835 (1GHz ARM11 core); 512 MB de LPDDR2 SDRAM; 1 slot de cartão micro-SD, 1 soquete mini-HDMI para saída de vídeo 1080p60; 2 soquetes micro-USB um para dados e outro para energia; 1 conjunto de 40 pinos de chave GPIO; 1 chave para vídeo composto e dimensões de 65mm x 30mm x 5mm. O sistema operacional empregado foi Raspbian, variação do Linux Debian mantida pela Fundação Raspberry Pi (FOUNDATION, 2015). Na Figura 12 tem-se uma imagem do principal computador de placa única utilizado neste trabalho: Raspberry Pi modelo Zero.

Figura 12 : Raspberry Pi Zero (EVERPI, 2015).

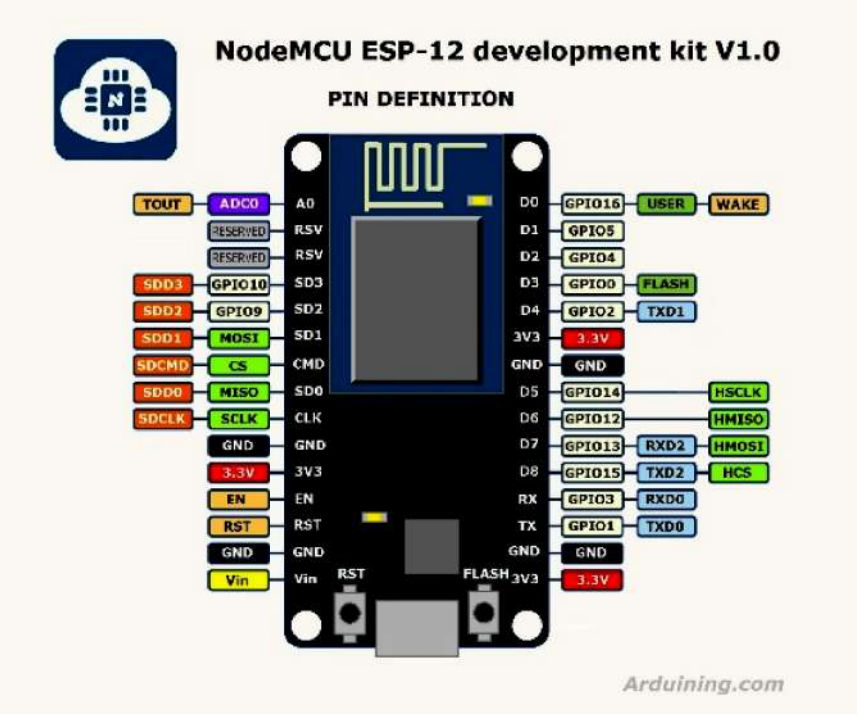


Cabe salientar que o experimento realizado nesta pesquisa utilizou ao todo dezessete computadores de placa única, sendo dez Raspberry Pi Zero, dois Raspberry Pi B versão 2, um Raspberry Pi B versão 3 e quatro Banana Pi M1. Todas essas variações do Raspberry Pi bem como seus clones foram utilizados para validar a aplicabilidade desta tecnologia (*openhardware single board computer*) em protótipo de Ambiente de Vida Assistida.

Ademais há a necessidade de se empregar vários dispositivos embarcados (*Networked Artifacts* SW/HW) para o papel de comunicação, sensoriamento e atuação no espaço AVA. Para tal o equipamento ESP8266 NodeMCU foi escolhido, com base em algumas recomendações encontradas em fóruns de desenvolvedores para Internet das Coisas (*Internet of Things*, IoT). Ademais, tem-se que um dos pressupostos deste trabalho é tornar economicamente viável a implantação de sistemas para AVA. Dessa forma, por ser um projeto *opensource* e contar com reduzido custo de investimento (menos de U$5) resolveu-se aplicar esta tecnologia (ESP8266 NodeMCU). Por fim, notou-se que o dispositivo necessitou de uma reduzida curva de aprendizagem para sua utilização (THOMSEN, 2016).

O artefato denominado ESP8266 NodeMCU é, portanto, uma placa de desenvolvimento IoT dotado apenas de: um chip ESP8266 com um antena Wifi Ethernet embutida, um firmware *opensource* NodeMCU, uma interface USB-serial e um regulador de tensão 3.3V.

Figura 13 : ESP8266 NodeMCU (THOMSEN, 2016).



A programação do NodeMCU pode ser feita usando diretamente a linguagem LUA como também usando a linguagem e IDE Arduino. Em acréscimo, o ESP8266 NodeMCU possui 11 saídas de E/S e um conversor analógico-digital para serem utilizados no sensoriamento e atuação no meio físico em que estiverem dispostos (NODEMCU-TEAM, 2017). Na Figura 13 tem-se o esquema de pinagem do dispositivo ESP8266 NodeMCU.

Cabe ressaltar que o experimento efetuado neste trabalho utilizou ao todo três dispositivos ESP8266 NodeMCU, sendo dois para sensoriamento (sensor de temperatura, umidade e luz) e dois para atuação (controle de luz e ventilador) em protótipo para AVA.

3.1.2.3. Variável dependente

Será utilizada apenas uma métrica como variável dependente: percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede sob implementações distintas (Algoritmo 1 e Algoritmo 2) do módulo detector de falhas em sistema para AVA. Cabe ressaltar que ambas as implementações efetuadas foram originalmente inspiradas no trabalho desenvolvido por Greve, Sens, *et al.* (2011) e em posterior atualização (GREVE, SENS, *et al.*, 2012).

3.1.2.3.1. Percentual de falsa suspeita de parada (percMistakes)

A estratégia utilizada para apuração do percentual de falsa suspeita de parada (percMistakes) foi a recepção das diversas listas de suspeitas de falhas ($susp_i$) da rede pelo nó líder do agrupamento. Com isso, torna-se possível estimar o percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede sob implementações distintas (Algoritmo 1 e Algoritmo 2) do módulo detector de falhas em sistema para AVA. A seguir, tem-se uma fórmula ilustrativa para o cálculo desse percentual:

$$percMistakes = \frac{quantidade\ de\ falsa\ suspeita\ de\ parada\ (crash)\ cometida\ na\ rede}{quantidade\ de\ nó(s) corretos(s)}$$

Cabe esclarecer que cada unidade do numerador da fórmula supracitada corresponde a pelo menos uma unidade de falsa suspeita de parada (definição 7) cometida por e em cada nó correto do sistema. Para fins de armazenamento, os diversos e possíveis cálculos de percentual de falsa suspeita de parada foram sucessivamente gravados a cada novo ciclo de avaliação do referido percentual em arquivo texto localizado no nó líder do agrupamento.

3.1.2.4. Tratamento

A seleção dos tratamentos aplicados no experimento deu-se através da execução independente de duas implementações do módulo de detecção de falhas: original (Algoritmo 1) e modificada (Algoritmo 2). Houve, portanto, o desligamento e posterior reinício dos nós sob cada uma das implementações (original e modificada) do detector de erros do sistema para Ambiente de Vida Assistida.

Ademais, utilizou-se a estratégia de eventos raros (Distribuição de Poisson[13]) para geração pseudoaleatória de falhas (inicialmente a parada de um nó). A falha resume-se ao desligamento lógico temporário (5 minutos) e intermitente[14] da interface de rede de um único nó. Este comportamento provoca a geração momentânea de suspeita(s) de parada (*crash*) na rede nos demais nós do sistema. Findo o lapso temporal de indisponibilidade do único nó problemático religa-se logicamente a interface de rede de modo que se inicia(m) a(s) possível(is) correção(ões) da(s) antiga(s) e agora falsa(s) suspeita(s) de parada sugerida(s) por qualquer outro processo da rede até a completa extinção desta(s) suspeita(s).

No fim, pretende-se que a lista de suspeitas de falha de todos os nós seja o máximo possível vazia ($susp_i = \{\}$), pois o roteiro de teste projetado visará garantir que nenhum nó permaneça indefinidamente parado durante toda a operação do experimento.

Conforme se verá adiante, verificou-se que este tratamento inicialmente adotado será insuficiente para o surgimento dos problemas identificados na fase de testes pós-desenvolvimento (subseção 2.3.3.3), fazendo emergir a necessidade de se incluir um dispositivo de comunicação (*swicth*) na estratégia de geração pseudoaleatória de falhas.

3.1.3. Projeto do experimento

Projetou-se o experimento de forma que a execução do roteiro de teste promovesse de forma isonômica a comparação da eficácia (grau de acurácia eventualmente fraca) obtida entre as duas implementações distintas do módulo de detecção de falhas em sistema para AVA: original (Algoritmo 1) e modificada (Algoritmo 2).

Ademais, optou-se pela constituição de um cenário verossímil (protótipo) de sistema

[13] A distribuição de Poisson indica que alguns eventos raros (ER) e aleatórios podem ser modelados. Ou seja, apesar de aleatórios, aqueles eventos não são completamente imprevisíveis, por mais que isto contrarie o senso comum. A distribuição de Poisson é usada para antever a chegada de eventos raros em servidores na Web, em centrais telefônicas e estações de *call centers*, entre várias outras aplicações da engenharia (ROSA, 2014).

14 A estratégia utilizada neste experimento para produção da pseudoaleatoriedade de falha em um nó através da execução infinita de um *script* que modifica temporariamente o estado de funcionamento da interface de rede em apenas um nó da rede cuja repetição (eventos raros) ocorre em intervalos crescentes em progressão aritmética cuja constante é 5 minutos.

para Ambiente de Vida Assistida.

3.1.3.1. Instrumentação

O processo de instrumentação foi efetuado a princípio com a configuração do ambiente experimental e, em seguida, o planejamento da coleta de dados.

O ambiente experimental foi constituído pelos seguintes equipamentos (*hardware*): computador de placa única (Raspberry Pi e clones), dispositivo embarcado (ESP8266 NodeMCU), fontes, cabos e *switches*, todos economicamente acessíveis no mercado e com baixo custo de investimento[15]. Da mesma forma, utilizaram-se *softwares opensource* para os componentes lógicos: sistema operacional, linguagem de programação, servidor de aplicação, plataforma AVA etc.

Pretendeu-se, com isto, comprovar a viabilidade de uso das tecnologias abertas (*open hardware/software*) para construção de um protótipo de Ambiente de Vida Assistida. Assim também, possibilitar a ampla repetição do experimento pelos eventuais interessados com vistas a comprovar os resultados alcançados.

Ademais, o código fonte das implementações original e modificada do módulo detector de falhas encontra-se disponível no site GITHUB através do seguinte endereço eletrônico: *https://github.com/Airton2Junior/Airton2Junior-fault_detector* (JESUS JUNIOR, 2017).

3.1.4. Operação do experimento

3.1.4.1. Preparação

A seguir, são enumeradas as etapas de preparação para a execução do experimento. Providenciou-se o mesmo ambiente computacional de *hardware* e *software* para ambas as implementações do módulo detector de falhas em análise. Seguem os passos executados para a fase preparatória do experimento:

a) primeiro, iniciou-se com a configuração física e, depois, a lógica dos dezessete computadores de placa única (*single-board computer*) de acordo com os requisitos exigidos pela plataforma AVA escolhida (OpenHAB mais Mosquitto);

b) em seguida, configurou-se fisicamente os computadores de placa única em *clusters* segundo tipo ou modelo: *cluster* I) dez Raspberry Pi Zero; *cluster* II) dois Raspberry Pi 2; *cluster* III) quatro Banana Pi M1 e, por fim, um Raspberry Pi 3 instalado

[15] O custo total de investimento deste trabalho não superou a importância de U$ 1000.00.

dentro do protótipo de Ambiente de Vida Assistida;

c) depois interligaram-se todos os dezessete nós do sistema (computadores de placa única) em uma rede LAN Ethernet (com e sem fio) sem a presença de qualquer tipo de falha em nós ou rompimento de enlaces;

d) ato contínuo, iniciou-se a execução do sistema, plataforma AVA munido do módulo detector de falhas desenvolvido, primeiramente sob a versão original do módulo detector de falhas, para avaliar se o ambiente computacional encontrava-se íntegro e apto para executar o roteiro de teste projetado.

e) por fim, desligaram-se todos os dezessete nós distribuídos do sistema.

Em resumo, o roteiro de teste projetado para o experimento exigirá a preparação física e lógica similar de todos os dezessete nós utilizados no experimento. Neste sentido, instalou-se a mesma plataforma AVA, OpenHAB mais Mosquitto, sob igual versão e configuração, além da mesma implementação do módulo detector de erros (incluindo o módulo de tolerância a falhas), de modo que cada tipo de implementação do módulo detector de falhas (original e modificada) possuísse as mesmas condições experimentais de operação.

3.1.4.2. Execução

Dado que todos os dezessete nós do experimento encontravam-se iniciados e que o sistema como um todo não apresentava qualquer tipo de falha, seja em nós ou enlaces, foi seguido o seguinte roteiro de teste:

1. primeiro, registra-se o início da Fase 1 que corresponde à execução do sistema inicialmente sob a implementação original do módulo detector de falhas (consoante Algoritmo 1);

2. passados novecentos ciclos de apuração de falsa suspeita de parada, que corresponde a aproximadamente a sete horas e meia de execução do sistema após o passo anterior, coleta-se todos os registros e percentuais de falsa suspeita de parada emitidos pelos diversos dispositivos que compõe o sistema para Ambiente de Vida Assistida – fim da Fase 1 do roteiro;

3. dado certo tempo, reconfigura-se todos os dispositivos do sistema, inclusive eliminado todos os possíveis vestígios de execução da Fase 1, para atuar agora

sob a versão modificada do módulo detector de falhas (consoante Algoritmo 2);

4. em seguida, registra-se o início da Fase 2 que corresponde à execução do sistema agora sob a implementação modificada do módulo detector de falhas;

5. decorridos outros tantos novecentos ciclos de apuração de falsa suspeita de parada, coleta-se todos os registros e percentuais de falsa suspeita de parada emitidos pelos diversos dispositivos que compõe o sistema para Ambiente de Vida Assistida – fim da Fase 2 do roteiro;

6. por fim, desligam-se todos os dispositivos empregados no experimento.

Para efeitos de validação do experimento foram efetuadas execuções anteriores e posteriores do roteiro de teste, com o propósito de avaliar se os diversos resultados encontrados difeririam substancialmente um do outro. Verificou-se, no fim, que os valores coletados em cada distinta execução do roteiro de teste apresentava-se resultados similares aos esboçados nos Gráficos 2 (Fase 1) e 3 (Fase 2).

Gráfico 2 : Apuração do percentual de falsa suspeita de parada em implementação original (Algoritmo 1) sob o tratamento inicial do experimento (falha pseudoaleatória em apenas um nó) – Fase 1

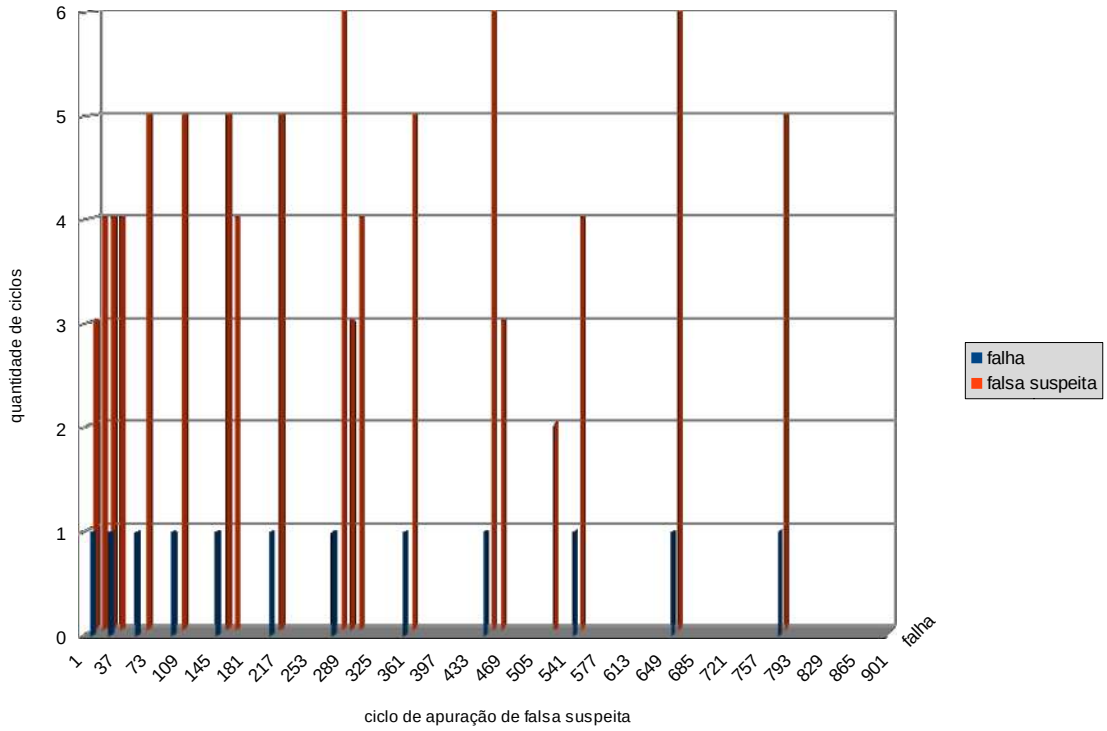
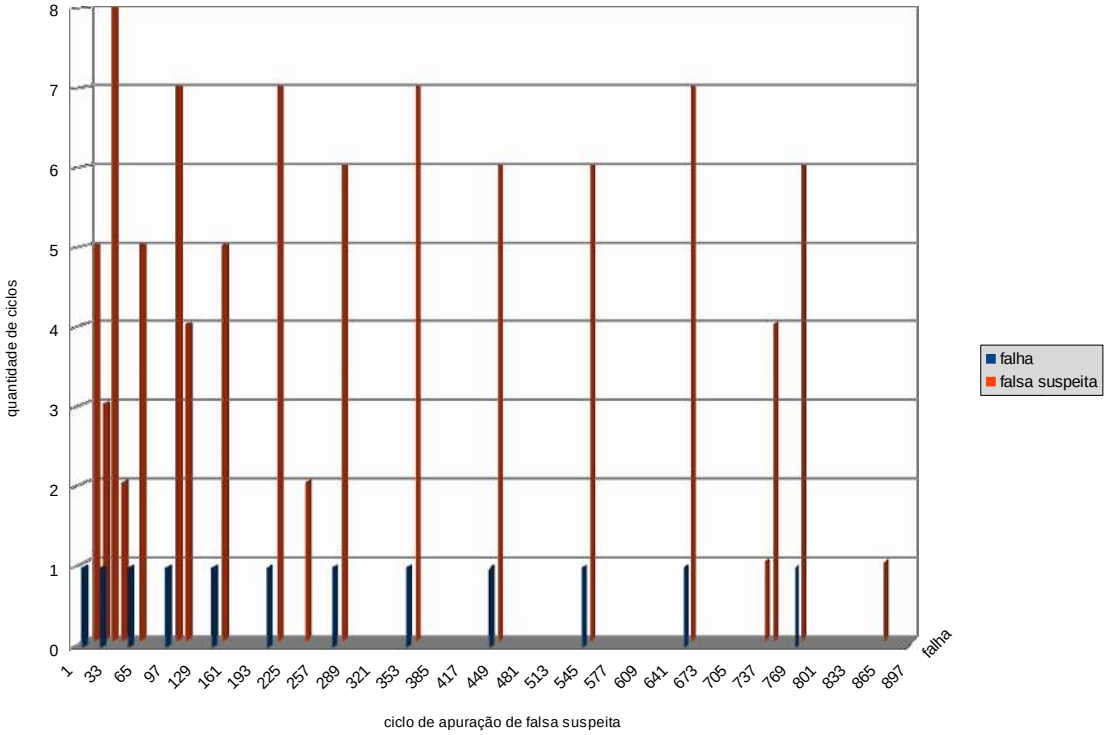


Gráfico 3 : Apuração de percentual de falsa suspeita de parada em implementação modificada (Algoritmo 2) sob o tratamento inicial do experimento (falha pseudoaleatória em apenas um nó) – Fase 2



Cabe destacar que o surgimento de suspeitas de parada na rede coincide em sua maioria com a geração pseudoaleatória de falhas projetada. Esta constatação pode ser observada através da frequente aproximação existente entre as quantidades de ciclos com falha (coluna azul) e falsa suspeita de parada (coluna vermelha) representadas nos Gráficos 2 e 3.

Ao final, verificou-se que o roteiro de teste tal como se encontra não foi suficiente para provocar o surgimento dos erros denotados em fase de testes pós-desenvolvimento (subseção 2.3.3.3). Desse modo, conforme antes mencionado, fez-se necessário incluir o elemento central de comunicação (*switch*) que interliga os três *clusters* de computadores de placa única e o protótipo de AVA na estratégia de geração de falhas pseudoaleatórias.

Sob esse novo tratamento refez-se o roteiro de teste e em dado momento surgiram os erros identificados na fase de pós-desenvolvimento do módulo de detecção de falhas em sistema para AVA (subseção 2.3.3.3). Os Gráficos 4 e 5 representam as quantidades de ciclos com falha (coluna azul) e falsa suspeita de parada (coluna vermelha) sob as implementações original (Algoritmo 1) e modificada (Algoritmo 2), respectivamente, sob esta novo tratamento de geração de falhas pseudoaleatórias em um nó e no *switch* central.

Gráfico 4 : Apuração do percentual de falsa suspeita de parada sob implementação original (Algoritmo 1) sob novo tratamento (falha pseudoaleatória em nó e switch) – Fase 1

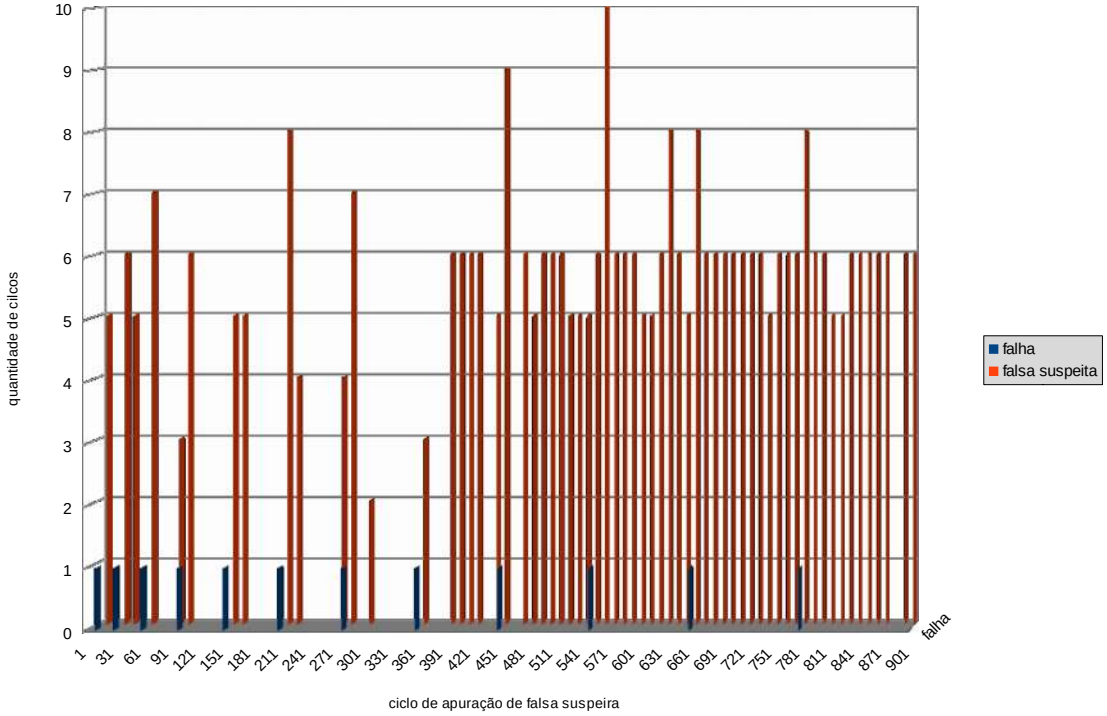
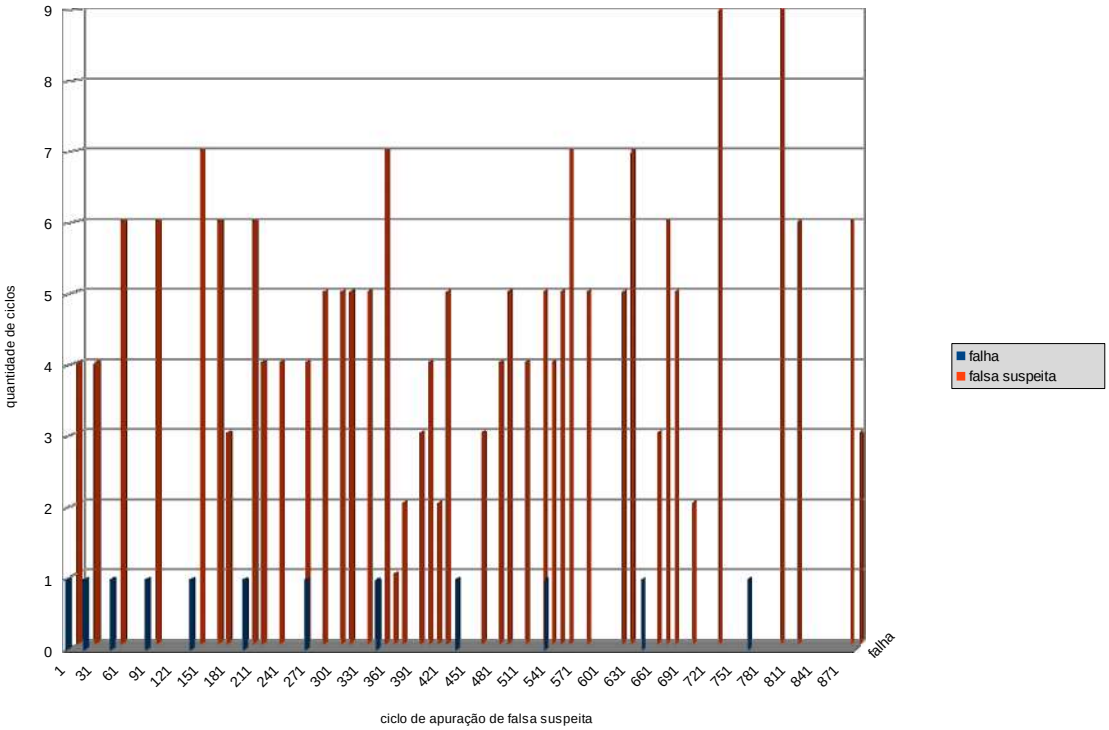


Gráfico 5 : Apuração do percentual de falsa suspeita de parada sob implementação modificada (Algoritmo 2) sob novo tratamento (falha pseudoaleatória em nó e switch) – Fase 2



Enfim, após as diversas execuções do experimento aplicou-se definitivamente o roteiro de teste considerando o segundo tratamento (falha pseudoaleatória em nó e switch), copiou-se o arquivo com os resultados alcançados no experimento para uma área de transferência, tendo em vista a necessidade de se efetuar uma análise estatística dos registros capturados.

Concluída a cópia de todos os registros de percentual de falsa suspeita de parada, desligou-se o sistema por meio da desativação total de todos os equipamentos.

3.1.4.2.1. Coleta de dados

A coleta de dados se deu através da cópia do arquivo texto contendo novecentos registros com percentual de falsa suspeita de parada (percMistakes) apurada pelo nó líder do agrupamento em cada fase do experimento (1 e 2).

3.1.4.2.2. Validação dos dados

Para realização do experimento, foi considerado um fator, percentual de falsa suspeita de parada (percMistakes) na rede sob duas implementações (original e modificada) do módulo detector de falhas, e um tratamento, geração de falhas pseudoaleatórias em um nó e no elemento central de comunicação (*switch*) do sistema. Diante disto, coletaram-se do nó líder de agrupamento os registros de percentual de falsa suspeita de parada (percMistakes) na rede durante todo roteiro de teste estabelecido no experimento. Como auxílio para análise, interpretação e validação, podem ser utilizados três tipos de testes estatísticos: Teste Kolmogorov-Smirnov[16], Teste T e o Teste U de Mann-Whitney[17].

O Teste Kolmogorov-Smirnov será utilizado para verificar normalidade das amostras. O Teste T pode ser utilizado para comparar a média das amostras, caso estas sejam normais e paramétricas e, por fim, o Teste U de Mann-Whitney possivelmente será efetuado para comparar as médias se as amostras independentes não obtiverem a normalidade nos dados, verificando a magnitude da diferença (WOHLIN, RUNESON, *et al.*, 2000). Todos estes testes estatísticos serão realizados por meio do software estatístico IBM SPSS™.

Por fim, para validar experimentalmente a proposta apresentada neste estudo fez-se necessário construir um protótipo de AVA com o emprego dos dispositivos antes apresentados. Na próxima seção descrever-se-á os elementos constituintes deste ambiente.

16 O teste de ajustamento de Kolmogorov-Smirnov destina-se a averiguar se uma amostra pode ser considerada como proveniente de uma população com uma determinada distribuição, no caso em questão Normal (Testes não paramétricos, 2017).
17 O Teste U de Mann-Whitney é usado para testar se duas amostras independentes foram retiradas de populações com médias iguais. Esse teste é, portanto, uma alternativa para o Teste T para amostras independentes quando a amostra for pequena e/ou as pressuposições, exigidas pelo teste "t", estiverem seriamente comprometidas. (PEREIRA, 2017)

4. Protótipo de Ambiente de Vida Assistida

Consoante mencionado anteriormente, para comprovar a viabilidade da presente proposta efetuou-se a confecção de um protótipo de Ambiente de Vida Assistida. Este protótipo consiste numa miniatura de domicílio de uma pessoa assistida, produzido em madeira, no qual se fez o comissionamento de dispositivos inteligentes em rede.

Figura 14 : Protótipo de Ambiente de Vida Assistida

Na Figura 14 tem-se uma visão geral do protótipo de Ambiente de Vida Assistida. Os ambientes projetados dividem-se em sala, cozinha, quarto e banheiro. Na parte superior têm-se o quarto e o banheiro da casa. No quarto instalou-se um ventilador, uma luminária, uma câmera e um sensor de temperatura e umidade. No banheiro colocou-se uma luminária. Na parte inferior do teto instalaram-se os dispositivos inteligentes. Na sala e cozinha (parte inferior) colocaram-se luminárias inteligentes, sendo que a da cozinha reage automaticamente à ausência de luz natural, ou seja, acende à medida que percebe a ausência de luz natural. Todos estes recursos interagem entre si e são gerenciados pela plataforma AVA instalada no computador de placa única Raspberry Pi modelo nº 3 instalado dentro do

protótipo de AVA. Nas Figuras 15 a 17 tem-se em detalhe cada ambiente mencionado.

Figura 15 : Detalhe do quarto do protótipo de Ambiente de Vida Assistida

Figura 16 : Detalhe da sala e cozinha do protótipo de Ambiente de Vida Assistida

Figura 17 : Detalhe da instalação dos dispositivos inteligentes no protótipo de Ambiente de Vida Assistida

Adicionalmente, preparou-se um conjunto de três *clusters* (agrupamentos) de

computadores de placa única de modo a proporcionar um quantitativo total de dezessete processos distribuídos com vistas a favorecer a validação do módulo proposto de detecção de erros em Ambiente de Vida Assistida. Na Figura 18 vê-se à esquerda o *cluster* de doze Raspberry Pi Zero. À direita tem-se outros dois *clusters,* sendo o primeiro superior à direita, *cluster* de dois Raspberry Pi 2; segundo superior à direita, *cluster* de quatro Banana Pi M1.

O *cluster* inferior à direita, *cluster* de dois Orange Pi Zero e dois Orange Pi One, não foi utilizado no experimento final haja vista que este agrupamento mesmo sob diversas tentativas de preparação manifestou comportamento errático quanto a comunicação com a rede, impossibilitando, portanto, o uso deste *cluster* na execução do roteiro de teste projetado. A Figura 19 mostra em detalhe este *cluster* mencionado.

Figura 18 : Visão geral dos *clusters* de computadores de placa única para execução do experimento

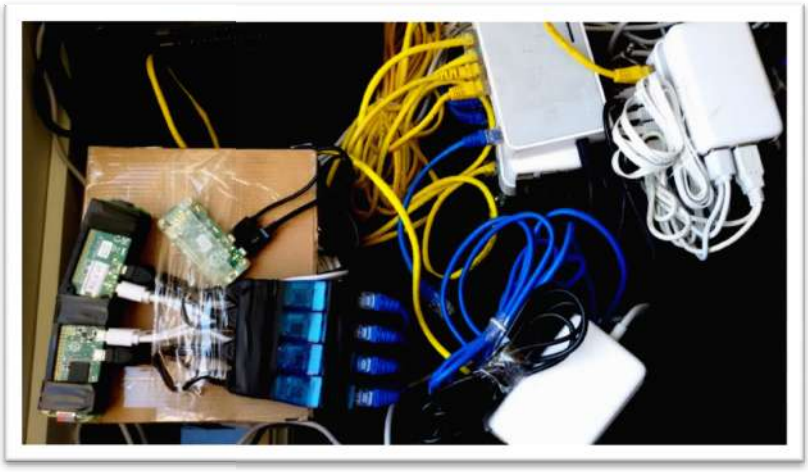


Figura 19 : Detalhe de um c*luster* de computadores de placa única para execução do experimento

5. Resultados

5.1. Análise e interpretação dos dados

Para responder à questão proposta na penúltima seção foi analisada a seguinte variável dependente: percentual de falsa suspeita de parada (percMistakes) quanto à definição da lista de suspeitas de falha ($susp_i$) na rede sob implementações distintas (Algoritmo 1 e Algoritmo 2) do módulo detector de falhas em sistema para AVA.

Um conjunto de percentuais de falsa suspeita de parada foi coletado pelo líder de agrupamento e armazenado em arquivo texto local para posterior análise e avaliação. Com estes registros efetuou-se o tratamento e os testes estatísticos dos dados coletados no experimento.

Na Fase 1, implementação original (Algoritmo 1) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e no elemento central de comunicação (*switch*) do sistema, tem-se como estatística descritiva o apresentado no Quadro 1.

Quadro 1 : Estatística descritiva da variável falsa suspeita de parada (percMistakes) em implementação original (Algoritmo 1) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e *switch* central – Fase 1

			Estatística	Erro Padrão
percMistakes	Média		38,05	1,579
	95% Intervalo de Confiança para Média	Limite inferior	34,95	
		Limite superior	41,15	
	5% da média aparada		36,72	
	Mediana		,00	
	Variância		2245,249	
	Desvio Padrão		47,384	
	Mínimo		0	
	Máximo		100	
	Amplitude		100	
	Amplitude interquartil		100	
	Assimetria		,492	,082
	Curtose		-1,718	,163

Para a Fase 2, implementação modificada (Algoritmo 2) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e no elemento central de comunicação (*switch*) do sistema, tem-se como estatística descritiva o apresentado no Quadro 2.

Quadro 2 : Estatística descritiva da variável falsa suspeita de parada (percMistakes) em implementação modificada (Algoritmo 2) do módulo detector de falhas sob o tratamento de geração de falhas pseudoaleatórias em um nó e *switch* central – Fase 2

			Estatística	Erro Padrão
percMistakes	Média		13,39	,996
	95% Intervalo de Confiança para Média	Limite inferior	11,44	
		Limite superior	15,35	
	5% da média aparada		9,32	
	Mediana		,00	
	Variância		892,775	
	Desvio Padrão		29,879	
	Mínimo		0	
	Máximo		100	
	Amplitude		100	
	Amplitude interquartil		0	
	Assimetria		2,117	,082
	Curtose		2,948	,163

Esses resultados sugerem que o percentual de falsa suspeita de parada (percMistakes), em média, na Fase 1 (implementação original) é superior ao percentual de falsa suspeita de parada (percMistakes), em média, encontrado na Fase 2 (implementação modificada) do módulo detector de falhas em sistema para AVA sob experimento.

Destarte, a partir da análise prévia destas informações, supõe-se que a resposta para a questão levantada na penúltima seção deste trabalho seja que a implementação modificada (Algoritmo 2) possui maior grau de eficácia (dado o fato de apresentar o menor percentual de falsa suspeita de parada) que a implementação original (Algoritmo 1) do módulo detector de falhas em sistema para Ambiente de Vida Assistida.

Entretanto, não é possível fazer tal afirmação sem evidências estatísticas suficientemente conclusivas. Para isto, primeiramente, definimos um nível de significância de 0,05 em todo o experimento e foi aplicado o teste Kolmogorov-Smirnov, para análise da distribuição normal.

O teste de normalidade da amostra coletada na Fase 1 (implementação original) foi avaliada pelo teste Kolmogorov-Smirnov conforme Quadro 3.

Quadro 3 : Teste de normalidade da Fase 1 (versão original – Algoritmo 1)

Testes de Normalidade[a]

	Kolmogorov-Smirnov[b]			Shapiro-Wilk		
	Estatística	gl	Sig.	Estatística	gl	Sig.
percMistakes	,378	900	,000	,645	900	,000

a. algoritmo = 1

b. Correlação de Significância de Lilliefors

Da mesma forma, o teste de normalidade da amostra coletada na Fase 2 (implementação modificada) foi avaliada pelo teste Kolmogorov-Smirnov conforme Quadro 4.

Quadro 4 : Teste de normalidade da Fase 2 (versão modificada – Algoritmo 2)

Testes de Normalidade[a]

	Kolmogorov-Smirnov[b]			Shapiro-Wilk		
	Estatística	gl	Sig.	Estatística	gl	Sig.
percMistakes	,453	900	,000	,498	900	,000

a. algoritmo = 2

b. Correlação de Significância de Lilliefors

Verifica-se, portanto, que o valor encontrado da variável Sig. 0,000 (zero), leia-se *p-value*, associado ao teste de Kolmogorov-Smirnov, é fortemente menor que o nível de significância adotado (0,05) para ambas as amostras em Fases 1 e 2 do experimento. Destarte, rejeita-se também a hipótese de normalidade (H_0) dos dados amostrais encontrados em ambas as Fases 1 e 2 do experimento.

Logo, assume-se que a distribuição dos dados, para ambas as amostras independentes, não é normal, e, portanto, devem ser avaliadas através de testes de hipóteses não paramétricos, a exemplo do Teste U de Mann-Whitney, exibido no Quadro 5.

Quadro 5 : Teste U de Mann-Whitney sobre as amostras independentes coletas na Fase 1 e 2 do experimento

Sumarização de Teste de Hipótese

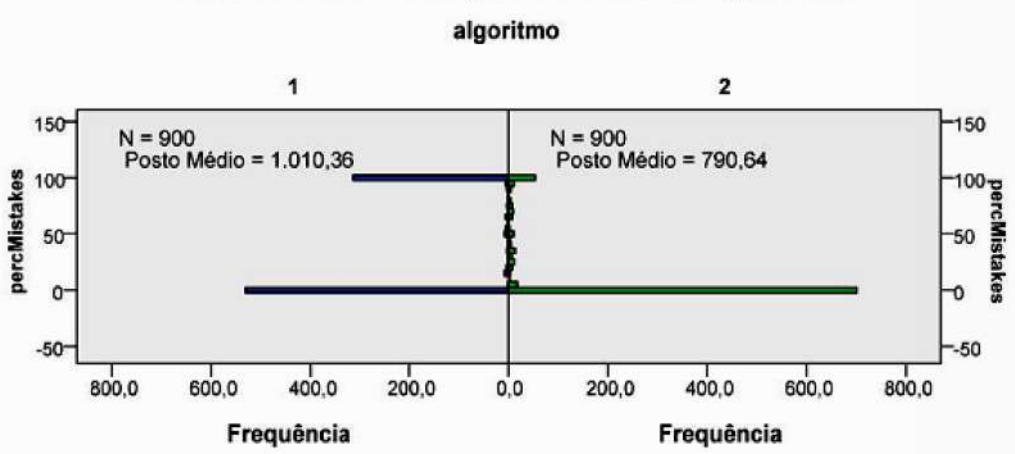
	Hipótese nula	Teste	Sig.	Decisão
1	A distribuição de percMistakes é a mesma entre as categorias de algoritmo.	Teste U de Mann-Whitney de amostras independentes	,000	Rejeitar a hipótese nula.

São exibidas significâncias assintóticas. O nível de significância é ,05.

Já no Quadro 6 tem-se em detalhe o resultado obtido pelo Teste U de Mann-Whitney.

Quadro 6 : Detalhe do Teste U de Mann-Whitney sobre as amostras independentes coletas na Fase 1 e 2 do experimento

Teste U de Mann-Whitney de amostras independentes



N total	1.800
U de Mann-Whitney	306.122,000
Wilcoxon W	711.572,000
Estatística do teste	306.122,000
Erro padrão	9.030,922
Estatística de Teste Padronizado	-10,949
Sig. assintótica (teste bilateral)	,000

Com a aplicação de testes de hipóteses não paramétricos, Teste U de Mann-Whitney, verificou-se que o Sig. de 0,000, obtido, é fortemente menor que o nível de significância adotado (0,05). Dessa forma, confirmou-se a evidência de diferença estatística entre as médias de percentual de falsa suspeita de parada (percMistakes) na rede obtido sob cada tipo de implementação do módulo detector de falhas, original na Fase 1 e modificada na Fase 2. Logo, rejeita-se a hipótese, a H_0 ($\mu_{\%Original} = \mu_{\%Modificada}$), e não se rejeita a hipótese alternativa, a H_1 ($\mu_{\%Original} > \mu_{\%Modificada}$).

5.2. Ameaças à validade

No que se refere à validade do experimento efetuado, tem-se as seguintes ameaças:

a) aleatoriedade na formação dos problemas de execução do Algoritmo 1 que propiciam a manutenção da falsa suspeita de parada na rede, tal como a autodeclaração ou mesmo o retardo na disseminação de suspeita consoante destacado na subseção 2.3.3.3;

b) utilização de diferentes tipos de computadores de placa única (*sigle board computer*) para representar os processos distribuídos p_i do experimento.

Quanto à primeira ameaça não se encontrou uma forma determinística de se delimitar o número de processos autodeclarados suspeitos e, portanto, "sob indevido engano", de modo a se restringir o caráter aleatório do fenômeno encontrado. Muito menos há como contornar o possível retardo na disseminação de mensagens QUERY entre nós vizinhos, visto que é intrínseco ao sistema para AVA a comunicação assíncrona entre os processos.

Ademais, quanto ao último item buscou-se utilizar padrões semelhantes de configuração de *hardware* e *software*, de modo a minimizar ou pelo menos impactar de modo similar os efeitos da execução do roteiro de teste em todos os dispositivos utilizados.

Cabe ressaltar que os problemas encontrados em subseção 2.3.3.3 apresentavam-se de forma similar e consistente nas diversas ocasiões em que, por ventura, apareceram durante as diversas execuções do roteiro de teste projetado neste trabalho. Os problemas apontados apresentavam-se mesmo quando se utilizava exclusivamente a participação de dispositivos físicos de mesma natureza e configuração.

6. Conclusão

De acordo com o delineado neste trabalho, considerou-se que dentre os erros mais comuns em sistemas, optou-se, por questões de simplificação, pelo erro de parada (*crash*) de processo. Dos paradigmas de detecção de erros em sistemas assíncronos do tipo *crash* escolheu-se o proposto por Greve, Sens, *et al.* (2011) visto ser o que mais se aproxima do contexto inerente a um sistema para Ambiente de Vida Assistida. Depois se desenvolveu um módulo de detecção de falhas e deste pretendeu-se verificar a eficácia na detecção de erros no contexto de sistema para Ambiente de Vida Assistida. Em seguida, montou-se um protótipo de AVA com o emprego de tecnologias abertas (*open hardware/software*) de modo a possibilitar a execução de um experimento controlado que validasse a proposta apresentada neste estudo.

Após a obtenção dos resultados experimentais, constatou-se, em primeiro lugar, a viabilidade da implantação de sistema tolerante a falhas para Ambiente de Vida Assistida com o emprego de tecnologias abertas ou economicamente acessíveis. Em seguida, verificou-se que a implementação de um detector de falhas no contexto de sistema para Ambiente de Vida Assistida, consoante originalmente proposto em Algoritmo 1 (GREVE, SENS, *et al.*, 2011), não satisfaz em grau máximo a acurácia eventualmente fraca especificada para a classe de detector de erros proposta, $\diamond S^{M}$.

Em virtude disso, efetuou-se algumas modificações (Algoritmo 2) e a partir destas constatou-se que os percentuais de falsa suspeita de parada (percMistakes) na rede são estatisticamente diferentes quando comparadas ambas as implementações (original e modificada) do módulo detector de falhas em sistema para AVA. Como também, não se encontrou instrumentos estatísticos que refutassem a hipótese alternativa do experimento, ou seja, que a média do percentual de falsa suspeita de parada (percMistakes) na rede sob a implementação original (Algoritmo 1) é superior a encontrada sob a versão modificada proposta pelo presente trabalho (Algoritmo 2).

No que se refere a trabalhos futuros, recomenda-se a pesquisa, o desenvolvimento e a avaliação de detectores de erros em sistemas para Ambiente de Vida Assistida sob outros modelos de falhas, tais como: bizantina e maliciosa, ou ainda, em situações que infrinjam políticas de segurança específicas etc.

Ademais, espera-se que estudos posteriores analisem e aperfeiçoem a estratégia (*gossiping*) e o desempenho do Algoritmo 2 proposto bem como mensurem e otimizem a taxa de consumo de energia demandada pelos dispositivos empregados no sistema para Ambiente de Vida Assistida com o emprego de tecnologias abertas (*open hardware/software*).

REFERÊNCIAS

AALIANCE2. European Next Generation Ambient Assisted Living Innovation Alliance, 07 nov. 2015. Disponivel em: <http://www.aaliance2.eu/>.

ANDERSON, T.; LEE, P. A. **Fault Tolerance Principles and Pratice**. Englewood Cliffs, NJ: Prentice Hall, 1981.

ANTONINO, P. O. et al. Evaluation of AAL platforms according to architecture-based quality attributes. **Ambient Intelligence**, Berlin Heidelberg, 2011. 264-274.

BASILI, V.; WEISS, D. A Methodology for Collecting Valid Software Engineering Data. **IEEE Transactions on Software Engineering**, 10 (3), November 1984. 728-738.

BIRMAN, K. Reliable Distributed Systems: Technologies, Web Services, and Applications, Berlim, 2005.

BRUCKNER, D.; YIN, G. Q.; FALTINGER, A. Relieved commissioning and human behavior detection in Ambient Assisted Living Systems. **e & i Elektrotechnik und Informationstechnik**, 129, 2012. 293-298.

CAO, J. et al. The eventual leadership in dynamic mobile networking environments. **13th Pacific Rim Intern. Symp. on Dependable Computing**, 2007. 123-130.

CHAIMOWICZ, F. A saúde dos idosos brasileiros às vésperas do século XXI: problemas, projeções e alternativas. **Rev Saúde Pública,** 31, n. 2, 1997. 184-200.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The Weakest Failure Detector for Solving Consensus. **Journal of the ACM (JACM)**, 43, n.4, 1996. 685-722.

CHANDRA, T.; TOUEG, S. Unreliable failure detectors for reliable distributed. **Journal of the ACM**, 43 v.2, 1996. 225–267.

CHETAN, S.; RANGANATHAN, A.; CAMPBELL, R. Towards fault tolerance pervasive computing. **Technology and Society Magazine**, 24, n. 1, 2005. 38-44.

DA SILVA FILHO, J. N. Doenças crônicas degenerativas, 2016. Disponivel em: <http://www.telehelp.com.br/telehelp/o-que-e.aspx>. Acesso em: 16 abr. 2016.

DE ANDRADE TARRINHO, P. M. T. **Confiança como factor de Segurança num Ambiente de Vida Assistido**. 2009. 75 f. Dissertação (Mestrado em Engenharia), Universidade do Minho, Minho. 2009..

DIMITROV, T. **Design and implementation of a home automation service gateway based on OSGi.** Düsseldorf: University of Duisburg-Essen, 2005.

EQUINOX, E. Eclipse Equinox, 20 jan. 2017. Disponivel em: <http://www.eclipse.org/equinox/>.

ERVATTI, L. R.; BORGES, G. M.; JARDIM, A. D. P. **Mudança Demográfica no Brasil no Início do Século XXI**. [S.l.]: IBGE - Instituto Brasileiro de Geografia e Estatística, 2015.

EVERPI. Raspberry Pi Zero: O computador de 5 dólares e 1Ghz! Conheça a nova versão da placa, 26 nov. 2015. Disponivel em: <http://blog.everpi.net/2015/11/raspberry-pi-zero-o-computador-de-5-dolares.html>.

FELICIANO, A. B.; DE MORAES, S. A. Demanda por doenças crônico-degenerativas entre adultos matriculados em uma unidade básica de saúde em São Carlos-SP. **Revista Latino-americana de enfermagem**, 7, n. 3, 1999. 41-47.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. **Journal of the ACM**, 32 v.2, 1985. 374–382.

FOUNDATION, R. P. RASPBERRY PI ZERO: THE $5 COMPUTER, 26 nov. 2015. Disponivel em: <https://www.raspberrypi.org/blog/raspberry-pi-zero/>.

FRIEDMAN, R.; TCHARNY, G. Evaluating failure detection in mobile ad-hoc networks, 1, n. 8, 2005.

FURFARI, F.; TAZARI, M.-R.; EISEMBERG, V. universAAL: an Open Platform and Reference Specification for Building AAL Systems. **ERCIM NEWS**, v. 87, p. 44-45, out. 2011. Disponivel em: <http://ercim-news.ercim.eu/en87/special/universaal-an-open-platform-and-reference-specification-for-building-aal-systems>.

GITHUB. openhab/openhab-distro contributors, 20 jan. 2017. Disponivel em: <https://github.com/openhab/openhab-distro/graphs/contributors>.

GITHUB. universAAL/platform, 20 jan. 2017. Disponivel em: <https://github.com/universAAL/platform/graphs/contributors>.

GREVE, F. et al. A failure detector for wireless networks with unknown membership. **European Conference on Parallel Processing**, Berlin Heidelberg, 2011. 27-38.

GREVE, F. et al. Eventually strong failure detector with unknown membership. **The Computer Journal**, 55, n. 12, 2012. 1507-1524.

HAILPERN, B.; SANTHANAM, P. Software debugging, testing, and verification. **IBM Systems Journal**, 41, n. 2, 2002. 4-12.

HYDRA, P., 23 nov. 2015. Disponivel em: <http://www.hydramiddleware.eu>.

JALOTE, P. **Fault tolerance in distributed systems**. New Jersey: Prentice-Hall, 1994.

JESUS JUNIOR, A. A. AAL Fault Detector. **GITHUB**, 2017. Disponivel em: <https://github.com/Airton2Junior/Airton2Junior-fault_detector>. Acesso em: 31 maio 2017.

JIMÉNEZ, E.; ARÉVALO, S.; FERNÁNDEZ, A. Implementing unreliable failure detectors with unknown membership. **Information Processing Letters**, 100, n. 2, 2006. 60-63.

LAPRIE, J.-C. Dependable computing and fault-tolerance: concepts and terminology. **Digest of Papers FTCS-15**, 1985. 2-11.

LLORET, J. et al. A smart communication architecture for ambient assisted living. **IEEE Communications Magazine**, 53, n. 1, 2015. 26-33.

MEMON, M. et al. Ambient assisted living healthcare frameworks, platforms, standards, and quality attributes. **Sensors**, 14, n. 3, 2014. 4312-4341.

MICROSOFT-DEVELOPER-NETWORK. Classe MessageQueue, 21 jan. 2017. Disponivel em: <https://msdn.microsoft.com/pt-br/library/system.messaging.messagequeue(v=vs.110).aspx>.

MOSQUITTO, P. E. Eclipse Mosquitto, 21 jan. 2017. Disponivel em: <http://projects.eclipse.org/projects/technology.mosquitto>.

MQTT. FAQ, 21 jan. 2017. Disponivel em: <http://mqtt.org/faq>.

MQTT. MQTT for Sensor Networks – MQTT-SN, 21 jan. 2017. Disponivel em: <http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn>.

NAKAGAWA, E. Y. et al. Relevance and perspectives of AAL in Brazil. **Journal of Systems and Software**, 86, n. 4, 2013. 985-996.

NEHMER, J. et al. Living assistance systems: an ambient intelligence approach. **Proceedings of the 28th international conference on Software engineering**, p. 43-50, 2006.

NODEMCU-TEAM. NodeMcu Connect Things EASY, 23 jan. 2017. Disponivel em: <http://www.nodemcu.com/index_en.html#fr_54745c8bd775ef4b99000011>.

OASIS-CONSORTIUM. OASIS MQTT Internet of Things Standard Now Approved by ISO/IEC JTC1. **OASIS**, 21 jan. 2017. Disponivel em: <https://www.oasis-open.org/news/pr/oasis-mqtt-internet-of-things-standard-now-approved-by-iso-iec-jtc1>.

OASIS-PROJECT. OASIS: quality of life for the elderly, 09 nov. 2015. Disponivel em: <http://www.oasis-project.eu/>.

OMRAN, A. R. **The epidemiologic transition in the Americas**. Washington: Pan American Health Organization, 1996.

OPENAAL. OpenAAL. the open source middleware for ambient-assisted living., 09 nov. 2015. Disponivel em: <http://openaal.org/>.

OPENHAB. Open HAB Architecture, 20 jan. 2017. Disponivel em: <http://www.openhab.org/features/architecture.html>.

OPENHAB. openHAB Introduction, 20 jan. 2017. Disponivel em: <http://www.openhab.org/features/introduction.html>.

OPENHAB, P. Welcome to openHAB, 19 jan. 2017. Disponivel em: <http://www.openhab.org/>.

OPENHAB, P. Welcome to the openHAB 2 Documentation, 20 jan. 2017. Disponivel em: <http://docs.openhab.org/>.

OPENHAB-COMMUNITY. Connecting distributed openhab instances, 20 jan. 2017. Disponivel em: <https://community.openhab.org/t/connecting-distributed-openhab-instances/3304/2>.

OPENHAB-FOUNDATION. openHAB Foundation, 20 jan. 2017. Disponivel em: <http://www.openhabfoundation.org/>.

OSGI, A. The Dynamic Module System for Java, 20 jan. 2017. Disponivel em: <https://www.osgi.org/>.

PAULO, D. "Infotainment", bom ou mau? **Circula Seguro**, 21 jan. 2017. Disponivel em: <http://www.circulaseguro.pt/condutor-e-ocupantes/infotainment>.

PEREIRA, V. C. M. TESTE U DE MANN- WHITNEY (TESTE U), 2017. Disponivel em: <http://www.leg.ufpr.br/lib/exe/fetch.php/disciplinas:ce001:vivian_-_teste_u_de_mann-whitney.pdf>. Acesso em: 31 maio 2017.

PIEPER, M.; ANTONA, M.; CORTÉS, U. Ambient assisted living. **Ercim News**, v. 87, 2011.

RASPBERRYPI-FOUNDATION. FAQ, p. https://www.raspberrypi.org/help/faqs/#introWhatIs, 23 jan. 2017. Disponivel em: <https://www.raspberrypi.org/help/faqs/#introWhatIs>.

ROBINSON, A. S. A User-oriented Perspective of Fault Tolerant System Models and Terminologies. **9th Internacional Symposium of Fault Tolerant Computing Systems**, 1982. 22-28.

RODRIGUES, G. N. et al. Dependability analysis in the ambient assisted living domain: An exploratory case study. **Journal of Systems and Software**, 85, n. 1, 2012. 112-131.

ROSA, N. B. A reincidência de eventos raros em empreendimentos inovadores-Modelagem pela distribuição de Poisson. **XXIV Seminário Nacional de Parques Tecnológicos e Incubadoras de empresas**, Belém, Setembro 2014. 20.

SANTOS, R. What is MQTT and How It Works, 29 jan. 2017. Disponivel em: <https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/>.

SENS, P. et al. Um Detector de Falhas Assıncrono para Redes Móveis e Auto-Organizáveis, Rio de Janeiro, RJ, Brasil, 2008. 931-944.

SOLER, V. et al. Domotic hardware infrastructure in PERSONA project. **Ambient Intelligence and Future Trends-International Symposium on Ambient Intelligence (ISAmI 2010)**, 2010. 149-155.

SRIDHAR, N. Decentralized local failure detection in dynamic distributed systems. **The 25th IEEE Symp. on Reliable Distributed Systems**, 2006. 143–154.

STEG, H. et al. **Europe is facing a demographic challenge Ambient Assisted Living offers solutions**. [S.l.]. 2006.

SUN, H. et al. The missing ones: Key ingredients towards effective ambient assisted living systems. **Journal of ambient intelligence and smart environments**, 2 n. 2, 2010. 109-120.

TAI, A.; TSO, K.; SANDERS, W. Cluster-based failure detection service for large-scale ad hoc wireless network applications. **Int. Conf. on Dependable Systems and Networks**, 2004. 805-814.

TANENBAUM, A. S.; VAN STEEN, M. **Sistemas distribuídos**. 2. ed. São Paulo: Pearson Prentice-Hall, 2007.

TAZARI, M. R. et al. The universAAL Reference Model for AAL. **Handbook of Ambient Assisted Living**, v. 11, p. 610-625, 2012.

TESTES não paramétricos, 2017. Disponivel em: <http://www.mat.uc.pt/~cmtm/ECwww/TestesNP.pdf>. Acesso em: 10 maio 2017.

THOMSEN, A. COMO PROGRAMAR O MÓDULO ESP8266 NODEMCU, 18 fev. 2016. Disponivel em: <http://blog.filipeflop.com/wireless/esp8266-nodemcu-como-programar.html>.

UNITED NATIONS. DEPARTMENT OF ECONOMIC. **World Population Prospects:** The 2015 Revision. New York: United Nations Publications, 2015.

UNIVERSAAL, P. The UniversAAL Reference Architecture, 2015 nov. 2015. Disponivel em: <http://www.universaal.org/images/stories/deliverables/D1.3-B.pdf>.

VOGELS, W. Tracking Service Availability in Long Running Business Activities. **Proc. First Int'l Conf. Service Oriented Comput. de Lect. Notes Comp. Sc. (Trento, Itália)**, Berlim, 2910, 2003. 95-408.

WHITTAKER, J. A. What is software testing? And why is it so hard? **Software**, 17, n. 1, 2000. 70-79.

WOHLIN, C. et al. **Experimentation in Software Engineering:** An introduction. [S.l.]: Kluwer Academic Publishers, 2000.

WU, W. et al. Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. **IEEE Trans. Comput.**, 8, 2007. 1055-1070.